CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

TRABAJO FIN DE GRADO

Desarrollo de una aplicación móvil iOS de información a estudiantes universitarios

Autor: Javier Alonso

Director: Xavier Ferré

MADRID, JUNIO DE 2014

Resumen

Reconociendo que podía ser útil para los alumnos una aplicación móvil para que pudieran acceder a información relacionada con sus estudios de forma rápida y sobre la marcha, se decide realizar una prueba de concepto cuyo resultado satisfactorio conduce a dar el siguiente paso en el desarrollo de la aplicación móvil. En este contexto es donde se enmarca el proyecto "Desarrollo de una aplicación móvil iOS de información a estudiantes universitarios" que tiene como finalidad aprovechar las ventajas que nos brindan las nuevas tecnologías.

En el prototipo de aplicación móvil fueron encontrados problemas de mantenibilidad y la versión del sistema operativo había quedado obsoleta. Por lo tanto el primer paso fue refactorizar todos los paquetes del proyecto, después de esto fue necesario crear un estándar de codificación y una documentación del proyecto. El segundo paso fue adaptar el proyecto a la última versión del sistema operativo, iOS 7, siguiendo la guía de transición de la interfaz de usuario de Apple.

Además de todo esto, había nuevas funcionalidades que incluir al prototipo, estas nuevas funcionalidades han sido probadas en una evaluación con usuarios para obtener comentarios y sugerencias de los alumnos universitarios para mejorar la aplicación en la medida de lo posible.

Recognizing that could be useful a mobile app for students to enable them to access information related to their studies quickly and on the go, it was decided to perform a proof of concept whose satisfactory results leads to take the next step in the development of the mobile app. In this context is where the project “Desarrollo de una aplicación móvil iOS de información a estudiantes universitarios” takes part whose aim is to exploit the advantages offered by the new technologies.

Maintenance problems were found in the mobile app prototype and the operating system version was outdated. So the first step was to refactor all the Project packages, after that it was necessary to create a coding standard and a Project documentation. The second step was to adapt the project to the latest versión of the operating system, iOS7, following the Apple UI transition guide.

In addition to all of this, there were new features to include to the prototype, these new features have been tested in an user evaluation to obtain feedback and suggestions from college students to improve the app as far as possible.

ÍNDICE DE CONTENIDO

1. INTRODUCCIÓN

La utilización de dispositivos móviles tipo *smartphone* por parte de los alumnos universitarios es cada vez mayor. En una encuesta realizada en el mes de Septiembre de 2012 a los alumnos de nuevo ingreso en la Facultad de Informática se ha podido comprobar que en torno al 90% de los alumnos utilizan un móvil de este tipo. De entre estos alumnos con *smartphone*, aproximadamente un 80% disponen de un móvil con sistema operativo Android o iOS.

Dado que los alumnos emplean el móvil habitualmente en su vida diaria, el hecho de utilizar esta vía como canal de comunicación adicional desde la Universidad puede aumentar el vínculo que los alumnos sienten con la UPM.

El alumno puede requerir en ciertas situaciones información sobre la marcha, sin posibilidad de acceso a un ordenador de sobremesa, y tener acceso rápidamente a dicha información a través del móvil le sería de gran utilidad. Así mismo, un centro universitario puede utilizar la posibilidad que tienen los móviles tipo *smartphone* de dar avisos, para distribuir noticias o anuncios de interés a los alumnos, complementando así el resto de canales de comunicación con el alumno (página web, correo electrónico, tablones físicos, etc.) y enriqueciendo de esta forma la gestión de la distribución de la información [1].

Promovidos por los motivos expuestos anteriormente se decidió implementar un prototipo como prueba de concepto de lo que podía ofrecer una aplicación de este tipo. El test de usabilidad que se realizó en Julio de 2013 con este prototipo con alumnos de la Escuela Técnica Superior de Ingenieros Informáticos (ETSIINF) permitió comprobar que sí se trataba de un proyecto viable que podía ser útil a la comunidad universitaria de la ETSIINF. Finalmente se decidió implementar la aplicación para las plataformas más utilizadas entre los estudiantes encuestados, es decir Android e iOS, siendo iOS la segunda plataforma con más usuarios entre los alumnos del centro.

En el presente documento se detalla desde un punto de vista de diseño software el trabajo realizado para desarrollar la aplicación móvil de la ETSIINF (a partir de ahora proyecto) de la UPM en la plataforma iOS, partiendo del prototipo existente con alguna de las funcionalidades previstas.

Dadas las características del proyecto, en el que son especialmente importantes la futura mantenibilidad y extensibilidad de la aplicación, el presente trabajo se centra en el rediseño de la funcionalidad existente y el desarrollo de nueva funcionalidad bajo unos criterios de diseño software que permitan realizar cambios de forma flexible en la aplicación. Así mismo, se sigue durante todo el proceso de desarrollo un enfoque de Diseño Centrado en el Usuario, aplicando las técnicas de Interacción Persona-Ordenador apropiadas y considerando las guías de diseño de la interfaz humana de Apple para la plataforma iOS [2].

1.1. Objetivos del proyecto

Partiendo de un prototipo de aplicación móvil de información a estudiantes de la ETSIINF de la UPM, el objetivo del proyecto era realizar el diseño, la implementación y las pruebas de una aplicación que extendiese el prototipo anterior mediante la inclusión de información relacionada con los títulos y las asignaturas, así como información de las líneas de autobuses o de contacto de las distintas unidades que componen la escuela, además de la adaptación del código existente a la versión iOS 7 del sistema operativo y la refactorización del código para una mejor mantenibilidad y extensibilidad de la aplicación.

1.2. Estructura del documento

El documento está estructurado de la siguiente manera:

- El presente capítulo es introductorio, y da una visión global de lo que se tratará a lo largo del documento.
- El segundo capítulo detalla los pasos necesarios para el desarrollo de aplicaciones en el sistema operativo móvil iOS.
- En el tercer capítulo se expone el prototipo de aplicación (prueba de concepto) previo y los problemas de diseño y mantenimiento encontrados.
- En el cuarto capítulo se detallan las tareas realizadas para adaptar el prototipo a la última versión disponible del sistema operativo.
- El quinto y sexto capítulo están dedicados al diseño que se le ha aplicado a nivel de software y de interacción a la aplicación móvil.
- En el séptimo capítulo se trata el proceso para posibilitar el multidioma dentro de la aplicación.
- El octavo capítulo se centra en las pruebas realizadas a distintos niveles por parte del desarrollador.
- En el noveno capítulo se detalla el proceso de evaluación de la primera versión de la aplicación por parte de los usuarios.
- El décimo y último capítulo contiene las conclusiones de todo el proceso de desarrollo del trabajo y las líneas futuras que podrían aportarse.

2. DESARROLLO DE APLICACIONES iOS

iOS es el sistema operativo de la empresa Apple Inc. desarrollado originalmente para el dispositivo móvil iPhone y también, con la posterior aparición de nuevos dispositivos, utilizado por el iPod Touch, el iPad y el Apple TV. La compañía no permite la instalación de este sistema operativo en hardware de terceros. La interfaz de usuario de iOS está basada en el concepto de manipulación directa, usando gestos multitáctiles.
En el desarrollo iOS se sigue el patrón de arquitectura de software Modelo-Vista-Controlador. Concretamente, todas las aplicaciones se basan en vistas. Cada una de las diferentes pantallas de una aplicación es una vista, y cada una de estas vistas se controla desde un controlador [3].

Como prerrequisitos para comenzar a desarrollar aplicaciones móviles en el sistema operativo iOS es necesario disponer de una computadora Mac con el kit de desarrollo de software (iOS SDK). El iOS SDK contiene el código, la información y las herramientas necesarias para desarrollar, probar, ejecutar, depurar y compilar aplicaciones para el sistema operativo iOS. Dentro de este kit encontramos tres aplicaciones fundamentales: el Xcode, el entorno de desarrollo integrado que facilita Apple en el que Objective-C es el lenguaje de programación en el que están escritas todas las aplicaciones, el Interface Builder, la herramienta gráfica que permite la creación de interfaces de usuario (disponible a partir de Xcode 4) aunque también se pueden crear por código otorgando más libertad al desarrollador y el iOS Simulator, el emulador del dispositivo en el que se pueden ejecutar las aplicaciones desarrolladas, también existe la posibilidad de ejecutarlas en un dispositivo móvil propio conectado, para esto último se necesita adquirir e instalar un certificado de desarrollador.
Cuando creamos un proyecto en el entorno de desarrollo Xcode, tenemos que configurar ciertas opciones que pasamos a describir:

- Product Name: Es el nombre de la aplicación que se va a crear.
- Organization Name: Es el nombre de la compañía o persona que está realizando la aplicación. El Xcode utiliza este nombre para diversos fines, tales como añadir una nota de *copyright* a cada fichero fuente.
- Company Identifier: Es una cadena única, que Xcode utiliza junto con el *product name* para crear el *bundle identifier* de la aplicación. Apple recomienda fuertemente la adopción de una convención de nombres en un intento de hacerlo único. Esta convención es nombrarlo como un dominio web pero a la inversa, por ejemplo *"es.upm.etsiinf".*
- Bundle Identifier: Este es el identificador de paquete, a pesar de que no se puede establecer esta opción de configuración en el momento de crear un nuevo proyecto de Xcode, se puede cambiar una vez creado. Por defecto este identificador es la combinación del *company identifier* y el *product name*. Hay que tener en cuenta que los espacios en blanco en el *product name* se reemplazan por guiones en este identificador.
- Class Prefix: Al crear clases personalizadas dentro del proyecto, es importante que los nombres de clase no colisionen con los nombres de clase existentes. Al especificar un prefijo de clase, Xcode lo asignará automáticamente en el momento de crear nuevas clases personalizadas evitando posibles colisiones.

- Devices: Esta opción de configuración le dice a Xcode el tipo de dispositivo, el cual va a ser objetivo de desarrollo.

La estructura de ficheros que vemos organizada de una manera dentro del proyecto, no es la misma que podemos ver en el Finder, la cual es mucho más simple. Mientras el directorio del proyecto utiliza una estructura de ficheros relativamente planos, la interfaz de Xcode usa un concepto conocido como "grupos" para organizar lógicamente los recursos del proyecto.
Por defecto, todos los ficheros de un nuevo proyecto iOS desarrollado con Xcode se estructuran en la interfaz del entorno de desarrollo según 4 grupos principales:

- La carpeta de nuestra aplicación**:** Es donde se encuentra el código fuente de la aplicación, se puede decir que es el proyecto como tal, esta carpeta lleva el nombre de nuestra aplicación. Tiene correspondencia física.
- La carpeta Tests: Es donde se encuentran disponibles herramientas de *testing* unitarias y el código destinado a la realización de pruebas concretas. Esta carpeta lleva el nombre de nuestra aplicación terminado en Tests, también tiene correspondencia física.
- La carpeta Frameworks**:** Esta carpeta contiene los *frameworks* básicos para poder desarrollar aplicaciones en iOS, no tiene correspondencia física.
- La carpeta Products**:** Esta carpeta contiene un archivo *.app* que se genera cuando realizamos una compilación, no tiene correspondencia física.

El entorno Xcode almacena la referencia lógica de la estructura de ficheros que hay debajo del proyecto y usa los grupos para ayudar a los desarrolladores a ordenar visualmente el entorno de desarrollo. Esta abstracción nos permite agrupar y/o recolocar los recursos del proyecto dentro de Xcode de forma sencilla sin que ello afecte a la estructura actual del proyecto en el disco. Sin embargo, como la representación física de un proyecto no es menos importante que la representación lógica del mismo y por consiguiente, debería también estar bien estructurada ya que a medida que vaya creciendo el proyecto y quisiéramos navegar por la estructura del código fuente ésta sería una tarea bastante tediosa. Una buena estrategia para crear una estructura de código fuente bien organizada es crear una estructura de ficheros acorde a las necesidades de nuestro proyecto y luego, importar estos cambios a nuestro proyecto en Xcode.

En el lenguaje de programación Objective-C, las clases se componen de dos ficheros (la interfaz, fichero .h y la implementación, fichero .m). Como se ha visto anteriormente todas las aplicaciones en iOS se basan en vistas, normalmente podemos decir que una vista se divide en 3 archivos. El de cabecera donde se hacen la definiciones de los métodos o funciones y/o propiedades de la vista, el de implementación donde se declara que hará cada método y donde se encuentra el ciclo de vida de la vista y el .xib (este fichero aparece cuando se utiliza la herramienta gráfica Interface Builder) que es la vista propiamente dicha.
Todas las clases tienen un ciclo de vida, para permitir al desarrollador manejar ese ciclo de vida existen un grupo de métodos por defecto, como el método *ViewDidLoad* que es el primero en ser llamado, estrictamente no lo es ya que el primero es el método constructor. Éste método se llama inmediatamente después de que la vista es cargada en la memoria, todo el código escrito en este método se ejecutará antes de que el usuario pueda ver la vista.

3. PUNTO DE PARTIDA: APP ETSIINF beta v.0.1

La app ETSIINF beta estaba desarrollada para la versión iOS 6 del sistema operativo de Apple y, dado que era un prototipo creado como prueba de concepto de aplicación móvil, no presentaba unas características de aplicación software profesional que aplicase el enfoque de la ingeniería del software. Éste es un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software.

3.1. Estructura de paquetes

Podemos decir que la organización general del proyecto no era la más apropiada. En términos generales la estructura del proyecto era confusa, en la cual el gran número de clases que la componen se encontraban en distintos paquetes sin seguir un patrón o modelo común en su agrupación.
La organización inicial de los paquetes del proyecto, refiriéndonos siempre a la organización dentro de la carpeta de nuestra aplicación que es donde se localiza el código fuente, era la siguiente:

FI UPM (carpeta de la aplicación)

- AppDelegate.h
- AppDelegate.m
- Clases
    - CollectionView
    - ManejadoresServicioWeb
    - Modelo de datos
    - Search Module
    - TableViewCustomCells
    - ViewControllers
- Ficheros Web
- Frameworks
- Imágenes
- Supporting Files

En esta organización nos podíamos encontrar con algunas clases, como la clase delegada de la aplicación, que se encontraban sueltas, sin agrupar dentro de ninguna carpeta, esta situación cobra más importancia en este caso por el hecho de ser Objective-C el lenguaje de programación en cuyas declaraciones de clases utiliza dos ficheros. Por tanto el número total de clases de un proyecto a medida que éste se extiende puede llegar a ser inmenso perjudicando a la ubicación.
El nombrado de las carpetas tampoco era correcto, se podían encontrar tanto carpetas cuyo nombre estaba en español como carpetas cuyo nombre estaba en ingles, siendo carpetas creadas por el desarrollador no por el entorno de desarrollo, lo que podía llevar a error. Por ejemplo la carpeta Frameworks, creada para albergar los *frameworks* propios, se confundía con la carpeta del mismo nombre creada por el entorno para albergar los básicos del sistema iOS.

3.2. Funcionalidades

Esta aplicación beta tenía implementadas las funcionalidades relativas al acceso a la información actualizada del tablón (Noticias, Anuncios, Eventos y Avisos), así como el acceso a la información del personal de la escuela (Departamentos y Servicios). Además de estas seis funcionalidades, incluía una funcionalidad de Mapa para poder localizar lugares de interés dentro de la escuela. Esta última funcionalidad se ha dejado apartada durante el periodo del presente trabajo por motivos relativos al cambio en la definición de la API utilizaba para la generación de los mapas.

Dentro del marco del presente trabajo se encontraba medio definido el servicio web que se iba a utilizar para las diferentes funcionalidades de la aplicación, estas eran Noticias, Anuncios, Eventos, Avisos, Servicios y Departamentos. El servicio web no estaba centralizado, estaba repartido por distintos servidores, no hay que olvidar que era un prototipo de aplicación móvil que no estaba conectado a los servidores del Centro de Cálculo.

A partir del presente trabajo la dirección del centro dio el siguiente paso en la producción de esta aplicación móvil, lo que ha conllevado tener un servicio web definido dentro de los servidores del centro en https://www.fi.upm.es/apps/para las seis funcionalidades previamente desarrolladas y para algunas de las que se han desarrollado durante este periodo de trabajo, como la de Autobús, Enlaces o Perfil, ésta última se encuentra aún en fase de implementación. También hay una funcionalidad más, la de Asignaturas, que se ha desarrollado en este periodo pero cuyo servicio web se encuentra alojado en un servidor del rectorado, en la dirección https://www.upm.es/gauss/api.upm/.

A continuación en la Figura 1 se muestra el diagrama del servicio web de la aplicación.

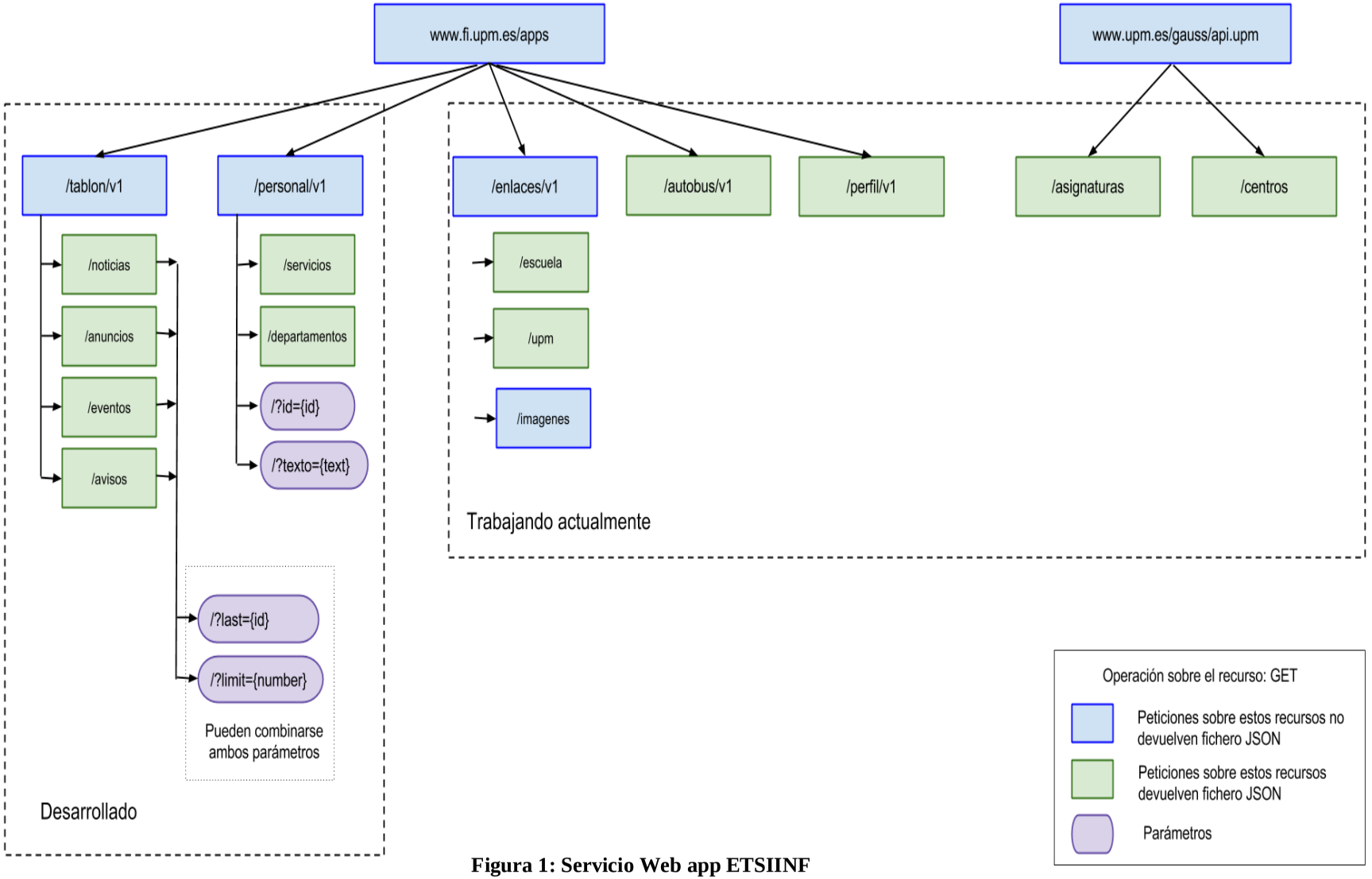

Figura 1: Servicio Web app ETSIINF

3.3. Nivel de presentación

Según se ha visto ya, en el desarrollo para iOS se sigue un patrón de desarrollo software Modelo-Vista-Controlador (MVC), pero este patrón no estaba bien aplicado en el prototipo inicial de aplicación móvil. La separación entre el Modelo y los Controladores era visible en la estructura de paquetes y acertada en su implementación, sin embargo entre los Controladores y la Vista no había ninguna separación. Cada una de las clases controladoras tenía a su vez el código para responder a los eventos y al ciclo de vida de la vista tanto como el código relativo a la creación de los elementos de la interfaz, por tanto todos los Controladores estaban mezclados con la Vista lo que implicaba problemas a la hora de cambiar la apariencia. Podemos decir que para cambiar una sola propiedad de apariencia de la Vista, tal como tipográfica o de color, a la hora de presentar el Modelo en un formato adecuado existían varios puntos de entrada. Sabiendo que el conjunto de la aplicación sigue el mismo diseño, debería de tener un único punto de entrada para modificar una propiedad gráfica y que surta efecto en toda la aplicación.

Otro problema importante a la hora de abordar el proyecto por parte de un nuevo desarrollador era que, por el hecho de no existir una documentación escrita de la estructura del proyecto, provocaba un proceso de adaptación largo. Por otro lado, los comentarios de documentación dentro del código fuente también eran inexistentes, por tanto la curva de aprendizaje era mayor.

Para el mantenimiento y revisión del código siempre hay que tener en cuenta que otros desarrolladores pueden continuar con el mismo, por lo que el problema de tener un código sin comentarios unido a la implementación según el modo y manera de cada desarrollador hace que el código resulte inmanejable.

3.4. Cambios a realizar

A continuación, después de analizar los problemas encontrados en la prueba de concepto, se toma la decisión de realizar una serie de cambios en este prototipo de aplicación para que se pueda considerar como aplicación software profesional. La proposición de cambios a realizar es la listada a continuación:

- Reestructurar todos los paquetes del proyecto.
- Modificar funcionalidades ya implementadas (especialmente en lo relativo al servicio web del Centro de Cálculo).
- Aplicar convenientemente el modelo MVC en todo el proyecto.
- Crear una documentación del proyecto.
- Crear un estándar de codificación (ver Anexo I).
- Añadir nueva funcionalidad.

4. ADAPTACIÓN iOS7

La última actualización del sistema operativo para dispositivos móviles de Apple iOS7, es la más revolucionaria del sistema iOS hasta la fecha. Se ha conseguido un rediseño casi total de la interfaz, aumentando la simplicidad y eficiencia, en palabras de la propia Apple. La puesta a punto de la interfaz la ha llevado a cabo Jonathan Ive, ésta es una interfaz más moderna, plana y translúcida en la que ha introducido un rediseño de los iconos, nuevos colores y una nueva tipografía: *Helvetica Neue*.
Teniendo en cuenta que actualmente la mayoría de los usuarios utilizan ya iOS 7, según información oficial proporcionada por Apple (ver Figura 2) y que la compañía tenía intención de lanzar en breve una beta de iOS 8 (finalmente presentada en Junio de 2014), se optó por dejar de dar soporte a iOS 6 y actualizar la aplicación completamente a iOS 7 [4].

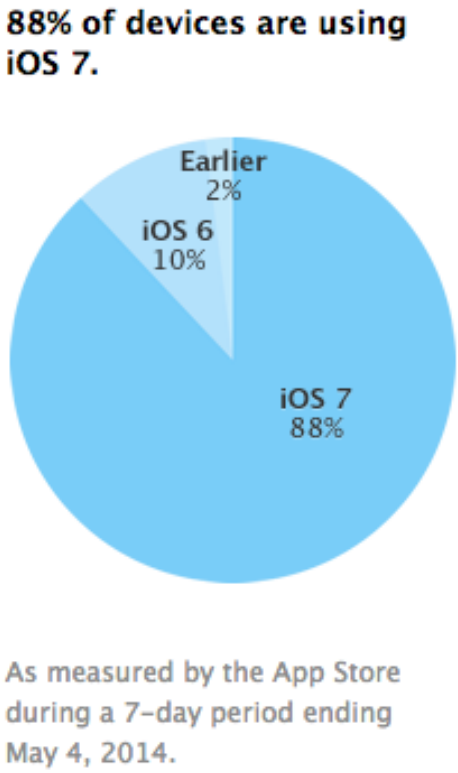

Figura 2: Gráfico de distribución

A pesar de que iOS 7 introduce muchos cambios de interfaz de usuario como botones sin borde, barras translúcidas y un diseño de pantalla completa para los controladores de las vistas, en algunos proyectos como en uno de los que nos podemos encontrar de ejemplo, llamado *TheElements,* las únicas diferencias dentro del entorno de desarrollo, a la hora de adaptar de la versión iOS 6 a la iOS 7, son el Deployment target (ahora 7.0) y el simulador.

Figura 3: Comparación de versiones

Como se puede observar en la Figura 3, a pesar de que todos los elementos de la interfaz se ven diferentes en iOS 7, los elementos de la API UIKit con los que se está familiarizado son fácilmente reconocibles. La librería UIKit proporciona todas las clases necesarias para construir y administrar la interfaz de usuario de una aplicación para iOS. Sólo en este tipo de aplicaciones que incluyen únicamente elementos estándar proporcionados por UIKit la adaptación no conlleva un gran esfuerzo, un caso distinto son las aplicaciones personalizadas, es decir las aplicaciones que no usan sólo los elementos de la UIKit.

En el caso de las aplicaciones personalizadas, como la del presente trabajo, en las que se quiere proporcionar una experiencia de usuario más satisfactoria, el proceso de adaptación es mayor. Según aconseja Apple en su guía de transición de interfaz de usuario [5], a la hora de realizar la transición a la nueva versión, el primer punto es centrarse en el rediseño de la aplicación para iOS7, en cuanto a cambios estructurales o de navegación.

Para poder conocer el alcance del proceso de adaptación de la aplicación se comprobaron en primer lugar las *checklists* proporcionadas por Apple para este propósito. Basándonos en estas listas concluimos realizar los siguientes pasos en el proceso de adaptación, explicados en las líneas que siguen:

Como primer paso, hubo que actualizar el icono de la aplicación. En iOS 7 los iconos de aplicación para las pantallas de alta resolución del iPhone son de 120x120 píxeles. Es importante notar que en IOS 7, siguiendo el estilo plano de la interfaz, no se aplican sombras ni brillos a los iconos de aplicación (ver Figura 4), por otro lado a estos nuevos iconos se les aplica una máscara de forma que redondea las esquinas reduciendo la curva para hacer una transición más suave con las líneas rectas, el radio de curvatura es diferente del que tenían los iconos de las versiones anteriores del sistema iOS. Finalmente el nuevo icono desarrollado para nuestra aplicación cumple estos requisitos de iOS 7(ver Figura 5).

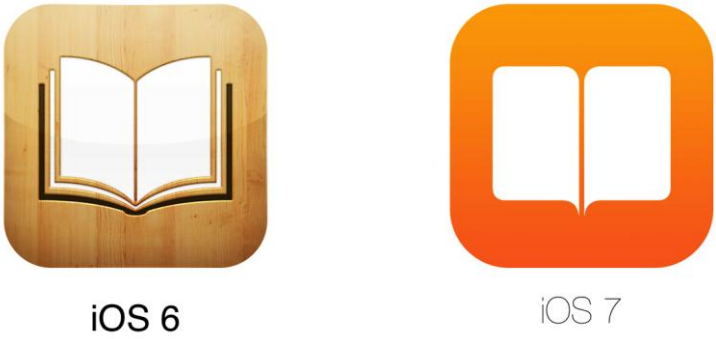

Figura 4: Iconos de aplicación

Figura 5: Icono app con nueva máscara

iOS define una gran cantidad de iconos estándar para las barras de botones, como el de Favoritos, Búsqueda y Ajustes, en la medida de los posible aconseja el uso de estos botones e iconos para representar las tareas estándar de la aplicación. Pero en aplicaciones como esta que incluyen tareas o modos que no pueden ser representados por estos iconos estándar, o simplemente porque estos iconos no coordinan con el estilo de la aplicación, se pueden diseñar iconos propios. Claro está que al mezclar en una aplicación iconos estándar e iconos personalizados

todos ellos deben parecer que pertenecen a la misma familia, en términos de tamaño percibido, nivel de detalle y apariencia visual.
Como en iOS 7 los iconos para la barra de botones son más ligeros en peso y tienen un estilo diferente, el siguiente paso fue rediseñar este tipo de iconos personalizados y sustituir uno de los iconos estándar de la colección de iOS por el nuevo modelo, en concreto el botón de acción, porque en esta versión ha cambiado (observar el cambio en la Figura 6).

Figura 6: Botones localizados en barras

Como en la versión iOS 7 la barra de estado es transparente, las medidas x (*top*) e y (*left*) no se toman a partir de la parte de abajo de esta barra, sino que se cogen de toda la pantalla, mostrando esta barra por encima de la vista de la aplicación. Según la guía de adaptación, teniendo en cuenta este cambio de la barra de estado, es necesario asegurarse de que el contenido de la aplicación se puede discernir a través de esta barra transparente y de otros elementos de la interfaz de usuario, tales como barras translúcidas y teclados.
Se realizaron las adaptaciones oportunas en el contenido de cada una de las vistas para, como se puede ver en la Figura 7, conseguir que el contenido fuese visible a través de elementos de interfaz translúcidos.

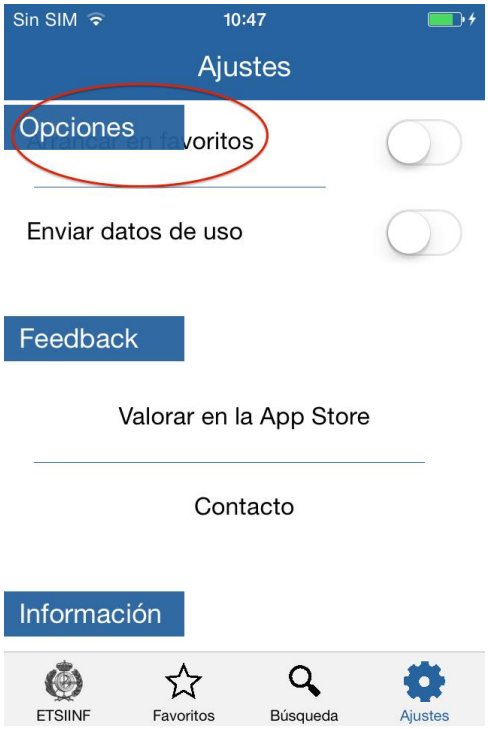

Figura 7: Elementos translúcidos

A causa de la nueva barra de estado transparente, y para seguir con el nuevo patrón dado por iOS 7, fue necesario adaptar la forma anterior de definir la barra de navegación en el prototipo a una nueva definición que incluyera el área que hay por debajo de esta nueva barra transparente. De acuerdo con la guía de adaptación y en relación con esta característica transparente, se recomienda actualizar la imagen de la pantalla de bienvenida de la aplicación. La versión beta de esta aplicación no disponía de este tipo de imagen, por lo que se optó por añadirla. En la parte superior de la Figura 8 se puede advertir la característica transparente de esta barra del sistema, en la que se muestran el estado de la conexión, la hora y la carga de la batería sobre el fondo azulado de la imagen de bienvenida.

Figura 8: Pantalla de bienvenida

Según recomendación de la guía, otro paso a tener en cuenta era examinar toda la aplicación en busca de los lugares donde las medidas y los cambios de estilo de los elementos UIKit en las vistas hubieran afectado al diseño y a la apariencia. Como ejemplo de cambio en un elemento del UIKit podemos poner los *switches*, que ahora son más anchos (ver Figura 9). En las versiones anteriores los botones tenían bordes, estaban recuadrados, los límites del botón se percibían mucho mejor, por tanto ahora la utilidad de las imágenes de los botones, usando menos borde, cobra más valor. A todo el diseño del prototipo de aplicación móvil se le aplicaba fondo gris con elementos en blanco, según el estilo común extendido entre las aplicaciones Apple de esas fechas. En versiones previas del sistema se jugaba mucho con el uso de sombras y degradados. Debido a que la estética iOS 7 es suave, con mucho menos énfasis en el uso de efectos visuales, hubo que volver a pensar en estos efectos. Estas características se pueden observar en la pantalla de inicio de la aplicación (ver Figura 10).

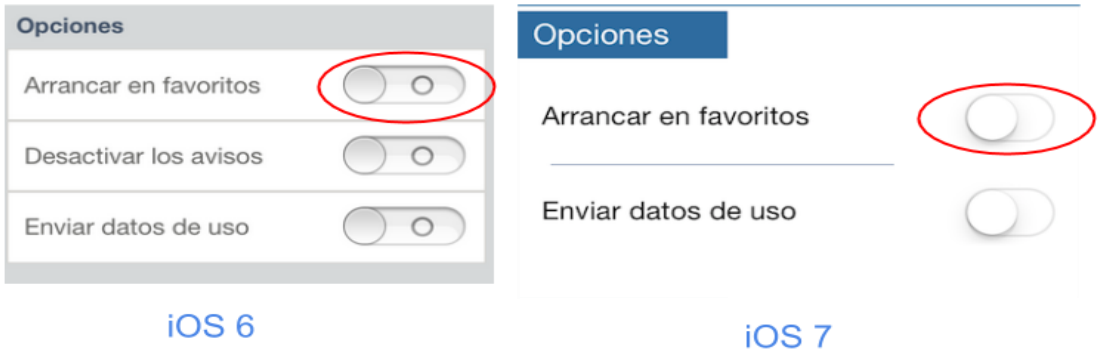

Figura 9: Elemento switch del UIKit

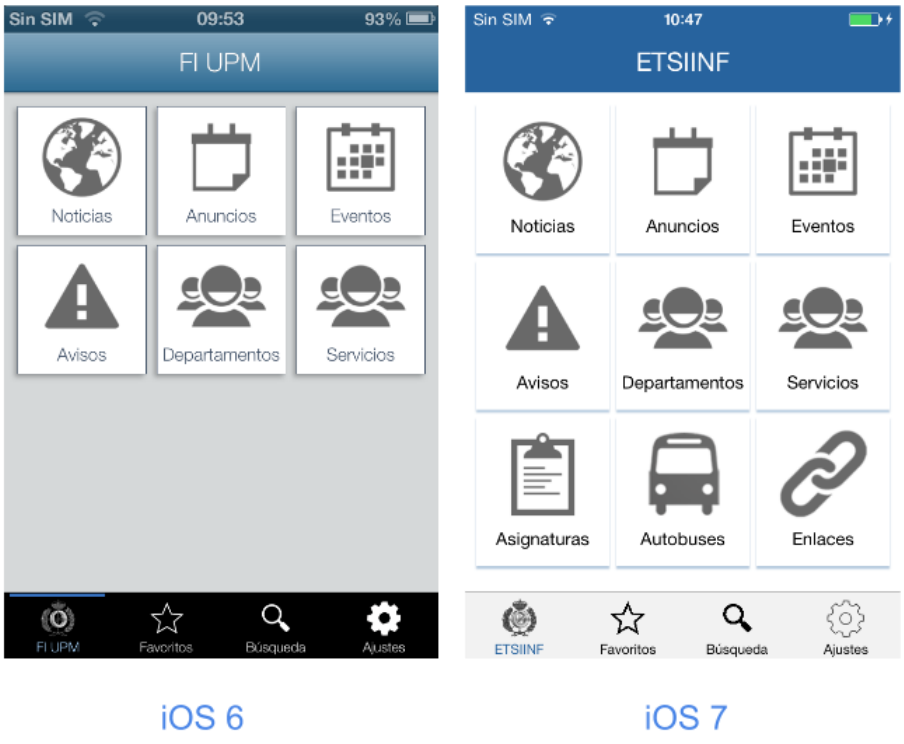

Figura 10: Pantalla de inicio

Como otra de las particularidades que se buscan con esta actualización del sistema iOS es eliminar márgenes, contenedores y bordes para que los elementos ocupen la totalidad del ancho disponible en pantalla, dentro de este proceso de adaptación, otra de las tareas fue redimensionar muchos elementos que habían quedado muy descolocados en pantalla. Las tablas por ejemplo eran elementos que flotaban sobre otra parte de la interfaz de usuario, sin embargo ocupando ahora todo el ancho se simplifica la tabla, dando una sensación de modularidad (ver Figura 11).

Figura 11: Tablas

Un paso importante fue el de revisar los valores *hard-coded* (datos incrustados directamente en el código fuente) de interfaz de usuario de toda la aplicación, tales como los tamaños y posiciones, y sustituirlos por valores calculados dinámicamente a partir de los valores proporcionados por el sistema.
En iOS 7 se ha añadido la propiedad Auto Layout para ayudar a la aplicación a responder ante cambios de diseño y no tener que estar sustituyendo valores, por este motivo en la guía de transición se aconseja la utilización de esta nueva propiedad. A pesar de no utilizar esta propiedad y tener que cambiar los valores a mano, la aplicación soporta perfectamente los tamaños de pantalla del iPhone 4S y anteriores y no habría ningún problema con pantallas de los dispositivos más actuales del mercado, a partir del iPhone 5.

En todas las versiones anteriores de iOS el tipo de fuente del sistema era *Helvetica*, en este prototipo ésta no era la fuente utilizada por toda la aplicación. Como iOS 7 ha introducido la fuente *Helvetica Neue* como fuente del sistema, éste nuevo tipo

de fuente es el que ha sido adaptado a toda la aplicación, sin dejar ninguna parte con un tipo de fuente diferente como ocurría anteriormente.

Algunas librerías quedaron obsoletas y hubo que quitarlas, como por ejemplo en el caso de una librería de notificaciones utilizada previamente por el prototipo, o actualizarlas, como en el caso de la librería de Google Analytics que sacó una versión especial para iOS 7. Como se comentó en el capítulo previo, el prototipo tenía una funcionalidad de Mapa que utilizaba la API de mapas de iOS, esta API cambió la definición con la nueva versión del sistema dejando inservible esta funcionalidad por lo que se decidió omitirla quedando anulada.
A su vez muchas referencias provenientes de librerías del sistema iOS 6 quedaron igualmente obsoletas teniendo que cambiar la definición de las declaraciones de muchos atributos y métodos.

En la quinta versión del entorno de desarrollo Xcode se ha cambiado la herramienta que incluía para realizar las pruebas unitarias del código fuente, por lo que las referencias existentes por defecto de la herramienta anterior, OCUnit, se han eliminado dando soporte a la nueva herramienta, XCTest (esta herramienta y el proceso de instalación se explicará detalladamente en el capítulo de pruebas).

Otra adaptación aplicada a los elementos pulsables, que en esta versión ya no disponen de bordes y son texto plano, ha sido poner un color común (azul) diferente al resto de texto plano (negro) para dar más *feedback* al usuario y que los pueda identificar fácilmente (ver Figura 12).

Figura 12: Elementos pulsables

Como último paso en la adaptación, se cambio el diseño del controlador de Detalle Persona para que, siguiendo la línea de diseño de Apple, se pareciese lo más posible a la aplicación de contactos del sistema. Se le aplicó un diseño teniendo siempre en mente que no llegara a ser completamente igual al de contactos para no dar al usuario la falsa sensación de que se encuentra en dicha aplicación en lugar de la presente. En la Figura 13 podemos detectar la similitud entre ambas.

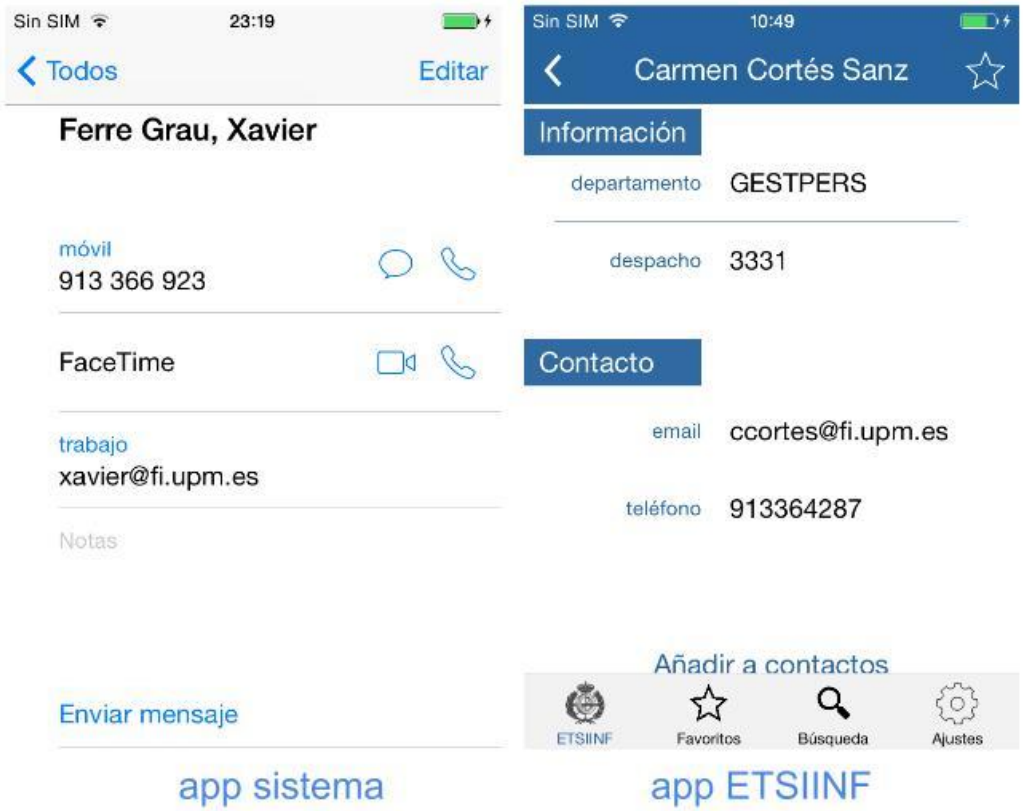

Figura 13: Pantalla contactos

Una vez comprobados todos los puntos de la *checklist* de Apple que afectaban directamente a la aplicación y que se tuvieron en cuenta para realizar esta tarea de adaptación, se pasó a mirar los puntos de la lista que no tuvieron efecto en el proceso de adaptación, como un consejo sobre la pantalla Retina o características nuevas que no incluía la versión previa de la aplicación. Estas características nuevas se tendrán en consideración para futuras versiones de la aplicación.

Un consejo que propone la guía de adaptación es tener en cuenta el soporte a la pantalla Retina en todo el diseño, este punto no supuso ninguna tarea de adaptación porque la aplicación ya lo había considerado previamente, incluyendo imágenes de distintos tamaños para las diferentes resoluciones de pantalla.

Ya que en la versión iOS 7 se ha incluido el Centro de Control, el cual aparece al deslizar el dedo hacia arriba desde la parte inferior de la pantalla. En la guía de adaptación se aconseja tener presente este hecho por si acaso la propia aplicación contiene un elemento pulsable en la parte inferior de la pantalla e interfiera este gesto con el normal funcionamiento de la aplicación. En nuestro caso los botones de la barra de herramientas son suficientemente grandes como para no suponer un problema de solapamiento al hacer el gesto de mostrar el Centro de Control. De todas formas, en las pruebas con usuarios se estudiará si hay problemas de este tipo.

Otra de las nuevas características que iOS 7 ha incluido y que nuestra aplicación no lo ha hecho es el Tipo Dinámico. Esto significa que los usuarios pueden ajustar el tamaño del texto que ven en las aplicaciones. Adoptando el Tipo Dinámico en la aplicación se obtiene un texto que responde adecuadamente a los cambios de tamaño especificados por el usuario.

Otra característica más, añadida por iOS 7, y que se tendrá en cuenta en futuras versiones para ayudar a la aplicación cuando se requieran cambios de diseño es la propiedad Auto Layout, nombrada previamente en este mismo capítulo.

5. DISEÑO SOFTWARE

En esta sección se detalla cómo se ha realizado el rediseño software de la aplicación móvil en su fase beta.
Hay que citar que la agrupación de las clases del proyecto en paquetes se ha realizado dentro de la carpeta de nuestra aplicación (llamada FI UPM).

5.1. Estructura de paquetes

A continuación vamos a estudiar el diagrama de paquetes de la aplicación para poder entender en detalle la relación que existe entre los diferentes paquetes de los que se compone nuestra aplicación.
La primera división de todos los ficheros de código fuente del proyecto se ha realizado en tres paquetes, el paquete AppNucleo, que reúne las principales clases de la aplicación móvil; el paquete Librerías con clases propias y de terceros empleadas para ciertas funcionalidades y por último el paquete Fuentes con los recursos utilizados como las imágenes para los iconos de la aplicación ó los ficheros JavaScript y CSS empleados en los elementos *webView* (ver Figura 14).

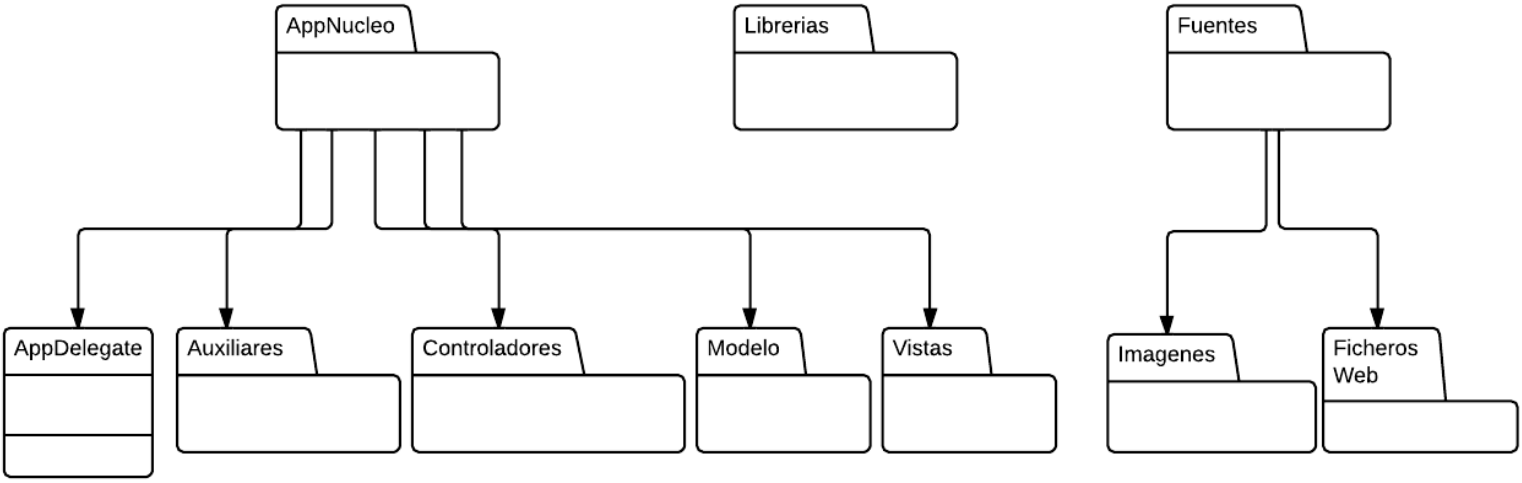

Figura 14: Aplicación móvil ETSIINF (Estructura)

5.2. Paquete AppNucleo

Dentro de este paquete se encuentra el núcleo principal de las clases de la aplicación, las que llevan toda la funcionalidad de la aplicación. La organización de las clases seguida en este paquete se ha regido teniendo en cuenta el patrón de arquitectura de software Modelo-Vista-Controlador que separa los datos, de la lógica y de la interfaz de usuario.

Por tanto se han organizado las clases según su función como Controlador (paquete Controladores), Modelo (paquete Modelos) ó Vista (paquete Vistas) y para el resto de clases cuya funcionalidad no encaja en ninguno de estos conceptos se ha utilizado el paquete Auxiliares, aparte de los paquetes anteriores se encuentra también el paquete con la clase delegada de la aplicación dejándola separada del resto.

A continuación se detalla en profundidad cada uno de los subpaquetes que componen este paquete:

5.2.1. AppDelegate

Paquete con una única clase, la clase delegada, encargada de la gestión de la aplicación. El AppDelegate maneja la aplicación como tal, define lo que se hace al abrir, al cerrar, al minimizar, al recibir una notificación, etc.
En esta clase es donde se añade el código personalizado para responder a los eventos dentro del ciclo de vida de cada ejecución de toda la aplicación.

5.2.2. Auxiliares

Paquete para las clases que, según su funcionalidad no se pueden encuadrar dentro del patrón MVC, como son los manejadores del servicio web, el módulo con métodos utilizados para realizar la búsqueda o el tema de la aplicación. Por tanto se han agrupado en un paquete aparte (ver Figura 15).

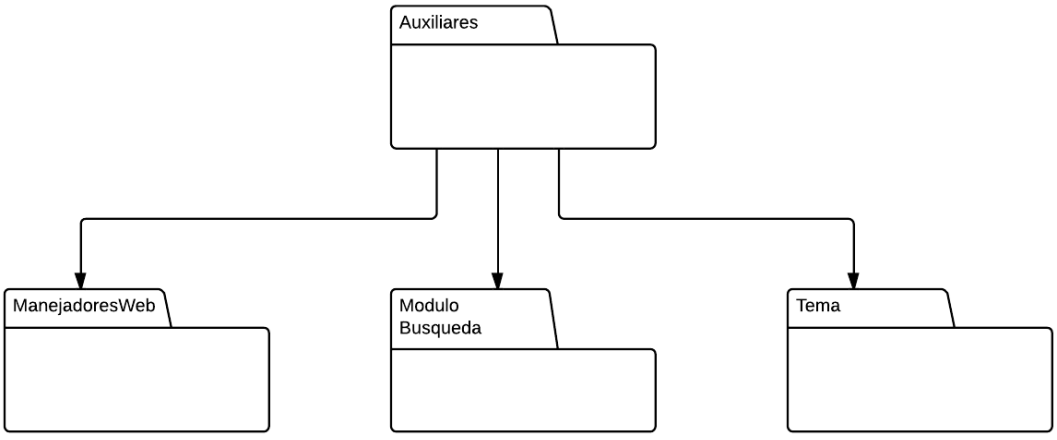

Figura 15: Paquete Auxiliares (AppNucleo)

5.2.3. Controladores

Clases utilizadas, según el patrón MVC, como controladoras de los elementos de la interfaz de usuario respondiendo a los distintos eventos (acciones del usuario).

Dentro de este paquete el razonamiento seguido para la organización de las clases ha sido colocar en paquetes separados los controladores asociados a cada uno de las diferentes secciones (*tabs*) de la aplicación. Esta decisión se ha tomado para que cualquier desarrollador pueda asociar fácilmente el código relacionado a cada sección (ver Figura 16), se puede observar que la sección Perfil no está disponible como funcionalidad en la versión actual de la aplicación, debido a que a la finalización de este trabajo todavía se encuentra en fase de implementación.
Dentro del paquete Principal, asociado a la pantalla de inicio de la aplicación, nos encontramos con una subestructura de paquetes en la que se encuentran los controladores asociados a las diferentes funcionalidades disponibles en cada uno de los botones de esta sección o pestaña, debido a la similitud entre algunas clases, se ha considerado agruparlas en un mismo paquete. Este caso se da en los paquetes Personal y Tablón. Dentro del paquete Personal nos encontramos con las clases dedicadas a las funcionalidades de Departamentos y Servicios; y dentro del paquete Tablón nos encontramos con las clases dedicadas a las funcionalidades de Noticias, Anuncios, Eventos y Avisos (ver Figura 17).
Adicionalmente, dentro de este paquete de Controladores, se encuentra un paquete con todos los Meta-controladores.

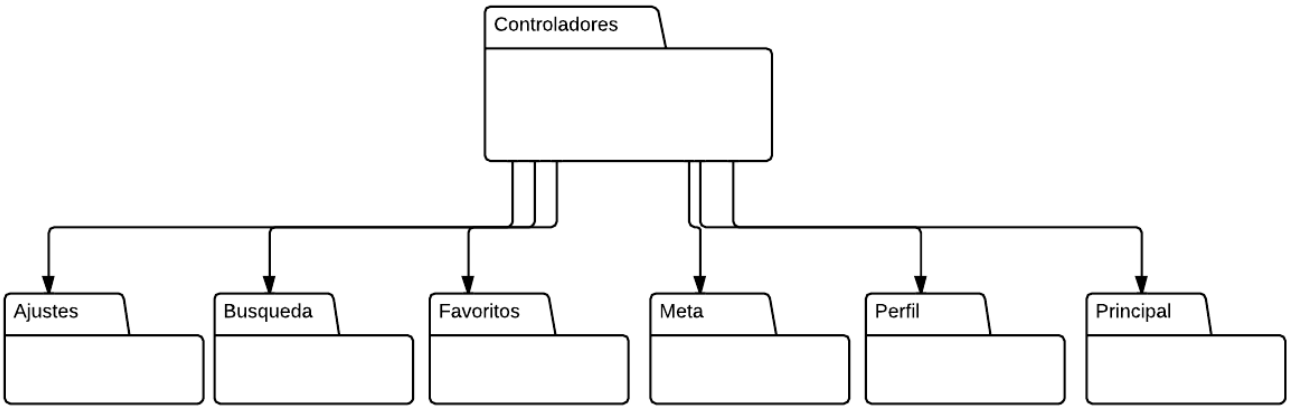

Figura 16: Paquete Controladores (AppNucleo)

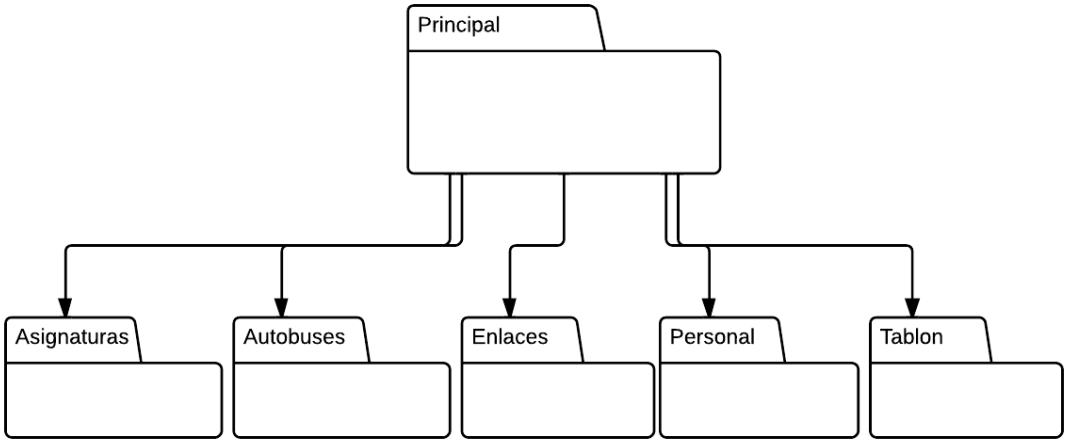

Figura 17: Paquete Principal (Controladores)

5.2.4. Modelo

Clases utilizadas, según el patrón MVC, como modelo de datos de la aplicación con toda la representación de la información con la cual trabaja la aplicación.
Debido a que la mayor parte de la información que maneja la aplicación está presente en la sección principal, nos encontramos con una subestructura de paquetes similar a la del paquete Principal, a excepción del paquete para el modelo de datos del apartado de Perfil (ver Figura 18).

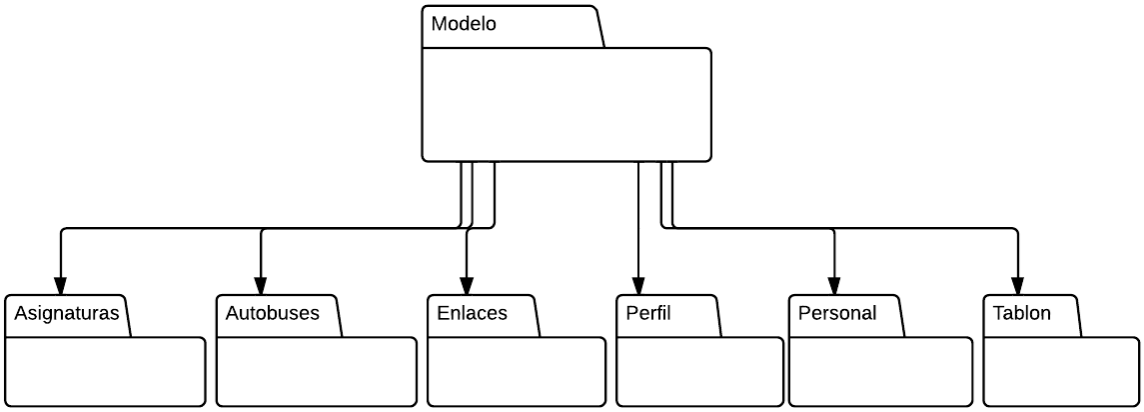

Figura 18: Paquete Modelo (AppNucleo)

5.2.5. Vistas

Clases utilizadas, según el patrón MVC, como elementos que presentan el modelo en un formato adecuado para interactuar. Aquí están definidos los elementos de la interfaz gráfica de usuario, elementos tales como un *collectionView*, un *tableView*, etc. Cada uno de estos elementos dispone de su propio paquete en el que se incluyen sus respectivos subelementos, como pueden ser sus celdas (ver Figura 19).

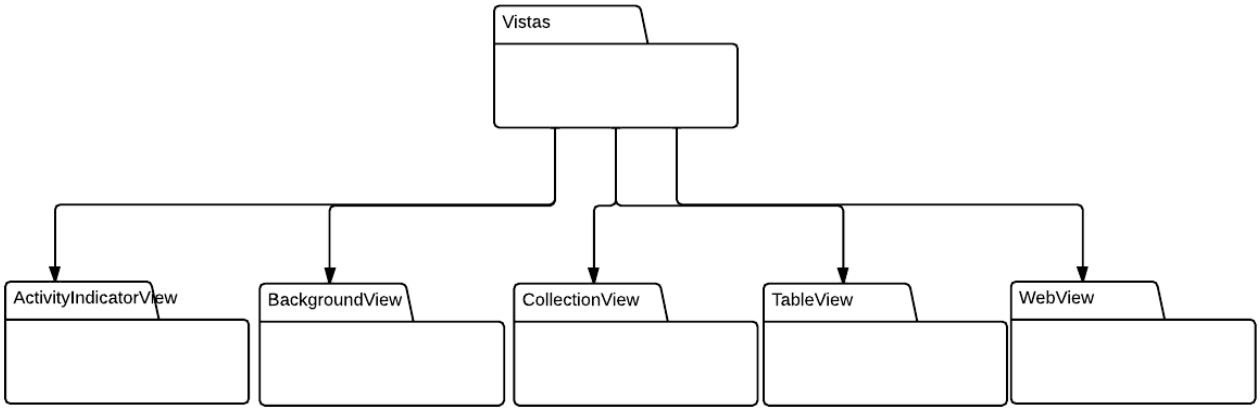

Figura 19: Paquete Vistas (AppNucleo)

5.3. Paquete Librerías

En este paquete nos encontramos los *frameworks* tanto propios, creados para algún propósito concreto, como desarrollados por terceros, para proporcionar alguna función específica como el de Google Analytics.

5.4. Paquete Fuentes

En este paquete nos encontramos con los tipos de recursos que utiliza la aplicación, pudiendo ser la colección de iconos de la aplicación dentro del paquete Imágenes ó los ficheros web usados en las *webViews* de las clases controladoras de detalle del Tablón dentro del paquete Ficheros Web.
Concretamente los ficheros que podemos encontrar dentro de Ficheros Web son: un fichero JavaScript y una hoja de estilos (fichero CSS). El *script* se utiliza para insertar la CSS en la web que se carga dentro del elemento *webView*. Del servicio web de la ETSIINF se recibe una página HTML que es exactamente la misma que se visualiza en la página web oficial, la cual tiene un CSS adaptado a navegadores web de escritorio, por tanto es necesario insertar un CSS propio adaptado a dispositivos móviles.

5.5. Supporting Files

Este no es un paquete como tal, creado por nosotros, es un grupo creado por defecto por el propio entorno de desarrollo Xcode, el cual no tiene correspondencia física en el disco.

Contiene archivos de ayuda y/o configuración de nuestra aplicación. Estos son:

- El archivo *.plist* contiene las configuraciones básicas, como nombre, identificador, orientaciones soportadas, etc.
- El archivo *.strings*, con este archivo podemos usar cadenas de texto "localizadas", podemos agregar idiomas y, usando el método correcto, reemplazar automáticamente los textos de nuestra aplicación según el idioma en el cual está el dispositivo. Por cada idioma tenemos una carpeta, *en.lproj* (idioma inglés) y *es.lproj* (idioma español).
- El *main.m* es el primer archivo que se ejecuta, este contiene la función principal de la aplicación, la cual realiza la primera llamada y donde se da inicio al ciclo de vida de la aplicación.
- El archivo *.pch*, en el cual podemos precargar clases y hacer definiciones, todo lo que definimos aquí estará disponible para todas las clases sin necesidad de incluirlas una a una. Es importante no abusar de este archivo ya que puede perjudicar el desempeño de la aplicación.
- Las imágenes bases: Son tres imágenes negras con distintos nombres. Estas imágenes son la imagen de la pantalla de bienvenida de la aplicación, la imagen que veremos mientras la aplicación hace las cargas básicas antes de poder ejecutarse. Hay tres imágenes porque el dispositivo iPhone tiene tres resoluciones posibles: No-Retina de 320x480 (imagen Default.png), Retina

de 640x960 (imagen Default@2x.png, exactamente el doble) y Retina de 4 pulgadas 640x1136 (imagen Default-568h@2x.png, la del iPhone 5).

5.6. Aplicación del MVC

A continuación se expone un ejemplo de la aplicación del modelo Modelo-Vista-Controlador a una de las funcionalidades de la sección Tablón, concretamente la de noticias. Este diagrama es aplicable exactamente, sin sufrir ninguna modificación, a las clases Anuncio, Evento, Perfil, Detalle y Enlaces.

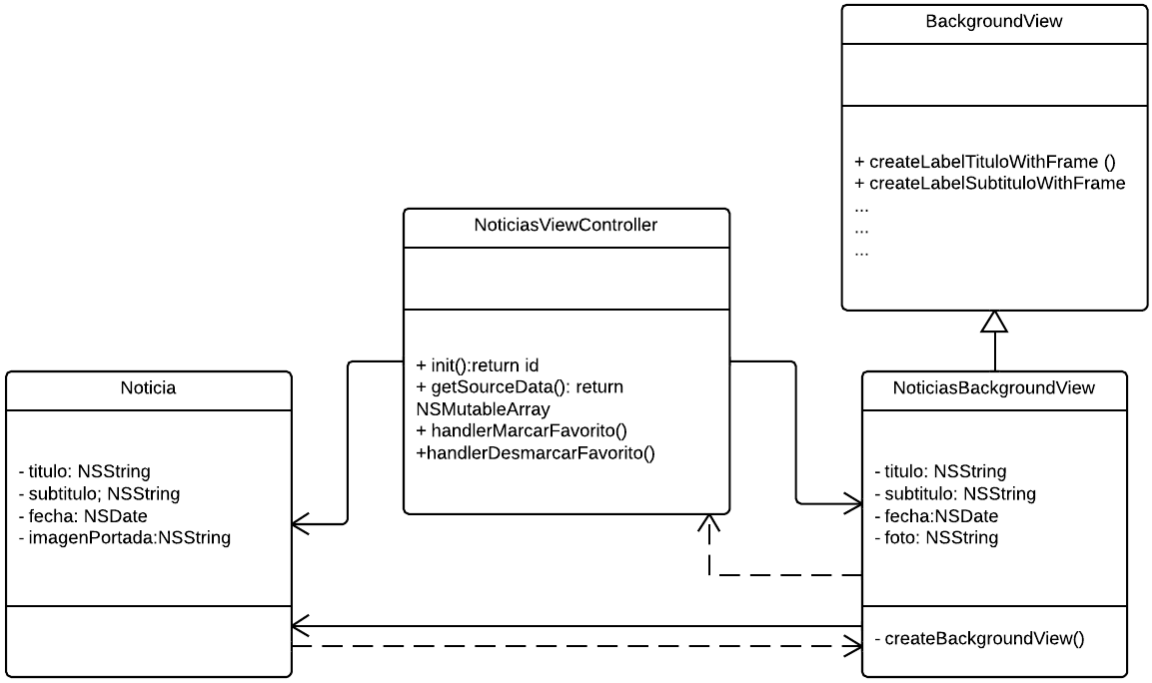

Figura 20: Diagrama UML de clases que siguen el MVC

El diagrama de la Figura 20 muestra la relación entre el modelo, la vista y el controlador. Las líneas sólidas indican una asociación directa, y las punteadas una indirecta (siguiendo un patrón Observador). En el diagrama tenemos la clase Noticia que pertenece al paquete del Modelo, la clase NoticiasViewController que pertenece a Controladores y la clase BackgroundView de la que hereda concretamente en este caso NoticiasBackgroundView, estas dos últimas clases pertenecen a Vistas.
El funcionamiento de este patrón en este caso concreto es el siguiente: El controlador, en primer lugar, descargará los datos de una noticia (titulo, subtitulo, fecha e imagen) a través del manejador del servicio web y se los asignará al modelo. Cuando el modelo ha cambiado su estado se lo notificará a la vista, de esta manera la vista tendrá constancia de que los datos están disponibles. Posteriormente el controlador creará una instancia de la vista, que en este caso es un elemento *backgroundView* con sus etiquetas correspondientes heredadas, y la cargará como un elemento más en la tabla de noticias que se muestra en la interfaz.

5.7. Modelo de datos

En este apartado se trata la modelización de los datos que maneja la aplicación, esta es toda la información asociada a la aplicación que se encuentra representada mediante las clases contenidas en el paquete Modelo. Para entender bien la relación de objetos presentes en cada modelo de datos del paquete, se ha elaborado un diagrama E/R por cada clase del paquete Modelo.

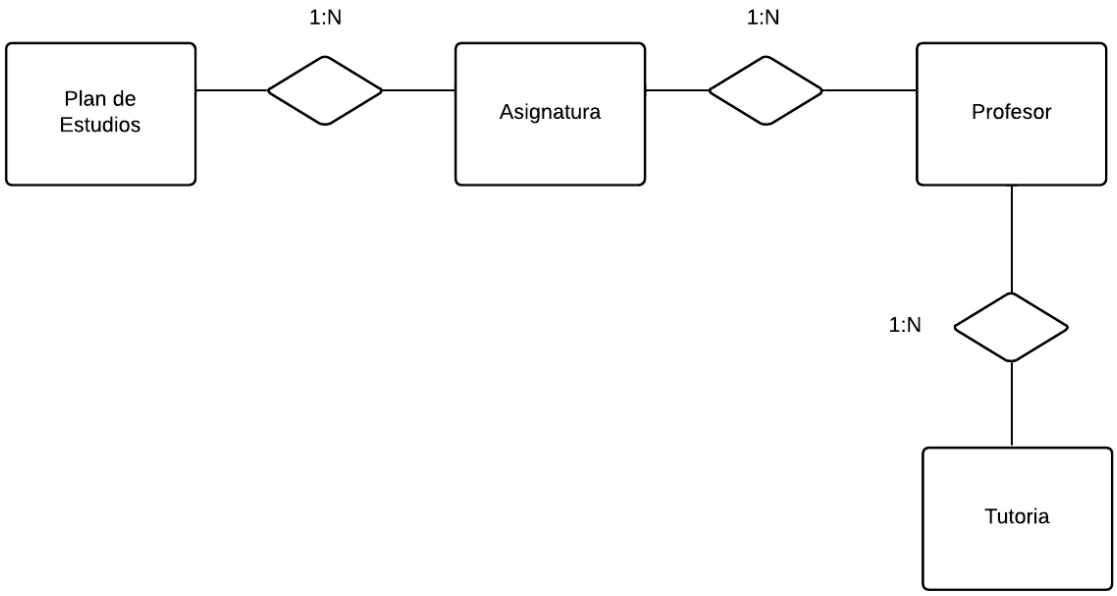

Figura 21: E/R Asignaturas

En la Figura 21 podemos apreciar que cada plan de estudios tiene varias asignaturas, cada una de ellas es impartida por varios profesores y estos tienen distintas tutorías a la semana.

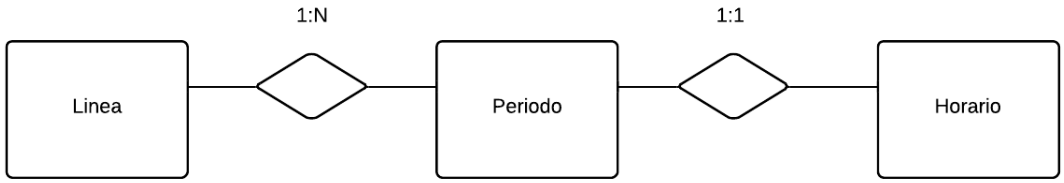

Figura 22: E/R Autobuses

En la Figura 22 podemos ver que cada línea de autobús tiene varios periodos (lectivo, no lectivo, Agosto, etc.) y en cada periodo hay diferentes horarios según el sentido.

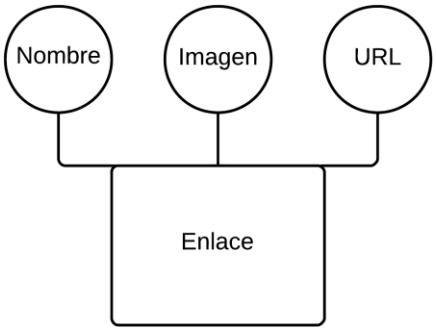

Figura 23: E/R Enlaces

En la Figura 23 podemos apreciar que la entidad enlace no tiene ninguna relación con otras entidades. Simplemente es una entidad que tiene un nombre de enlace, una imagen y la URL a la que apunta.

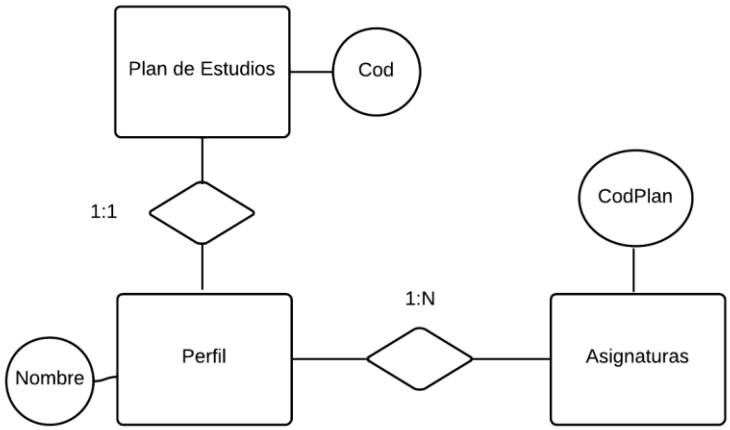

Figura 24: E/R Perfil

Se puede ver en la Figura 24 que el perfil de cada usuario está asociado a un plan de estudios en el que el usuario se ha matriculado de una o varias asignaturas, todas ellas correspondientes a ese plan.

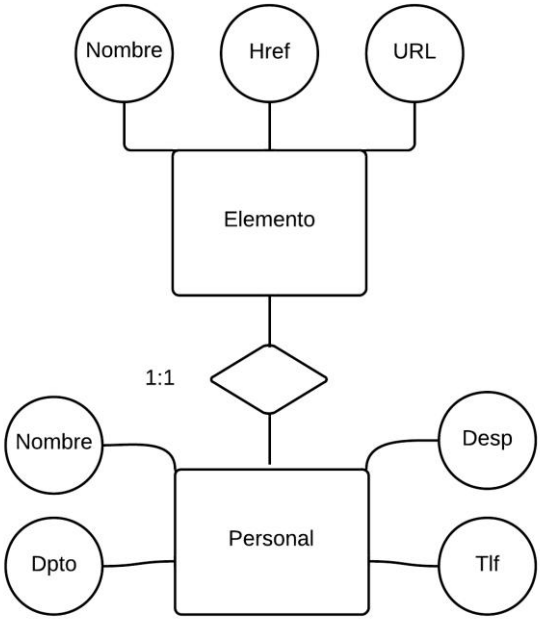

Figura 25: E/R Personal

En la Figura 25 podemos apreciar que toda entidad de personal, tanto sea profesor o de servicios, comparten los mismos atributos. Cada entidad de personal se relaciona con una entidad elemento para la lista de contactos.

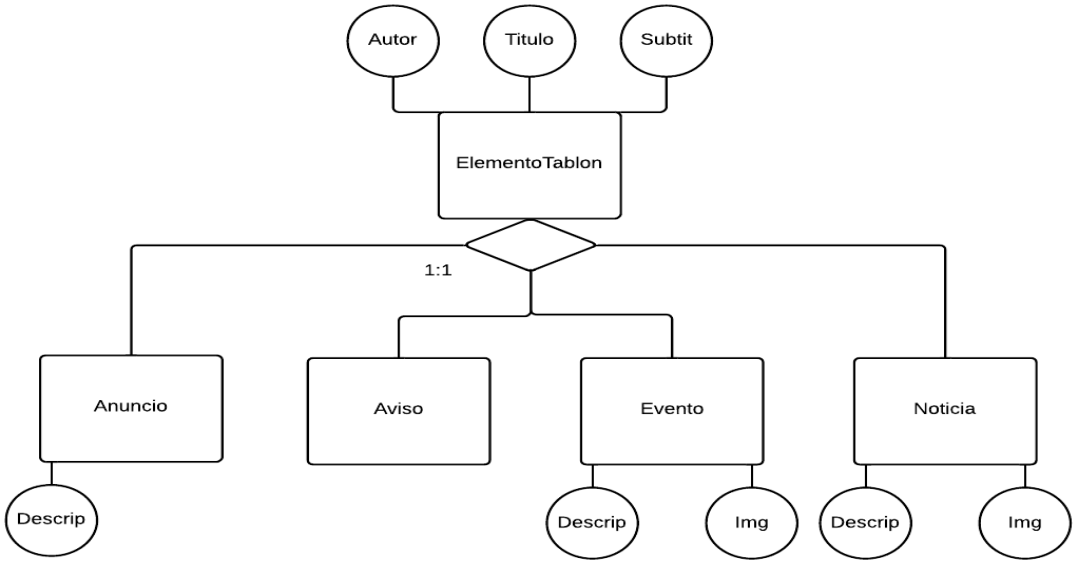

Figura 26: E/R Tablón

Podemos observar en la Figura 26 que cualquier entidad, ya sea de anuncio, aviso, evento o noticia es un elemento tablón con un autor, titulo y subtítulo común. Particularmente unas tienen atributo de descripción, imagen o ninguno de ellos.

6. DISEÑO DE LA INTERACCIÓN

La mayoría de las aplicaciones del iPhone tienen una estructura común con la que los usuarios del iPhone están acostumbrados y se sienten cómodos con ella. Por tanto para llevar a cabo un diseño de la interacción acorde con estas expectativas de los usuarios de iPhone, se consultaron las "Guías de interacción humana" de Apple [2] y el "Estudio sobre filosofía de interacción de iOS" presente en [6] y. Estos documentos se han tenido en cuenta para elaborar el mapa de navegación y el esquema de diseño de los distintos contextos de interacción que se detallan a continuación.

6.1. Mapa de navegación

A continuación, en la Figura 27, se presenta el mapa de navegación de la aplicación. Como se puede observar, la vista principal del mapa de navegación es la que presenta todas las funcionalidades de la aplicación y contiene una barra de pestañas con la que podemos pasar a los apartados de favoritos, búsqueda y ajustes. El mapa de navegación no tiene una profundidad excesiva, entre dos y tres niveles, decisión tomada al considerar que una profundidad mayor dificulta el entendimiento por parte del usuario del lugar/nivel de la aplicación en el que se encuentra. Por lo general en cada rama se muestra la información genérica y el detalle de esa información. Concretamente las ramas del tablón siguen ese patrón general de navegación con esos dos niveles, a excepción de las ramas de personal y asignaturas que añaden un nivel más con un listado de elementos. Mención aparte requieren las ramas de autobús y enlaces que disponen de un único nivel en el que se muestra contenido diferente mediante un control segmentado.
Como particularidad, y en relación con la profundidad del mapa de navegación, para no alcanzar un cuarto nivel en la rama de asignaturas se optó por incluir en el primer nivel un botón que muestra otra vista al mismo nivel con la que se selecciona y renueva el contenido de ese nivel en cada interacción con el mismo. Esta determinación se basó en el modo en el que se renueva el contenido en la AppStore. Finalmente también se incluyó este elemento en la rama de autobús para cambiar de línea.

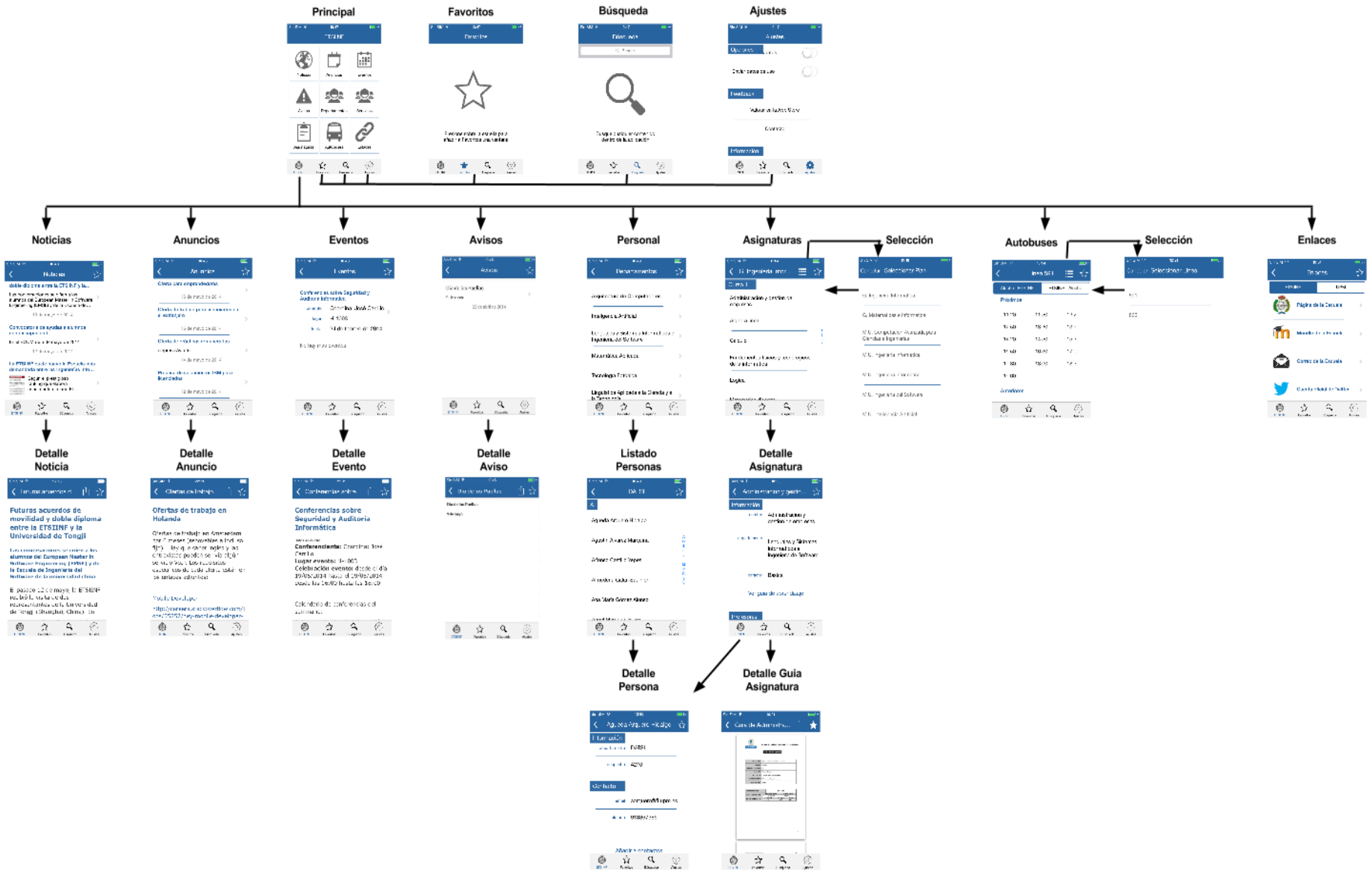

Figura 27: Mapa de navegación

6.2. Esquema de diseño

Teniendo en cuenta los patrones de diseño de la interacción en iOS, especialmente en iOS 7 ya que algunas decisiones del diseño de esta aplicación se tomaron en el momento de la prueba de concepto y hubo que adaptarlas como ya se ha comentado en un capítulo previo, se realizó el diseño de la aplicación basándose en un esquema general para todas las vistas y en el caso de algunas vistas en concreto el esquema general se ve modificado en la zona de contenido.
Este esquema de diseño de la aplicación puede verse en la Figura 28.

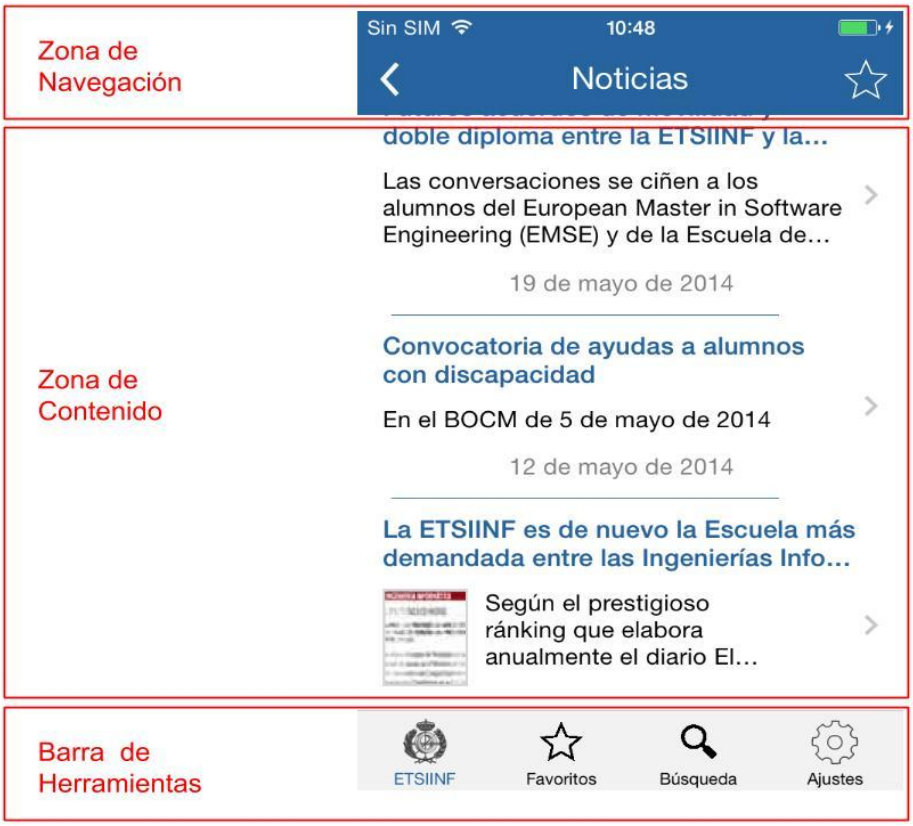

Figura 28: Esquema de diseño

Para esta aplicación, en la zona de navegación se ha decidido incluir el nombre de la ventana en la que nos encontramos y un botón en la parte derecha que se encargará de añadir esa vista a los favoritos, adicionalmente a este botón aparece un segundo botón en las vistas de Asignaturas y Autobuses para categorizar el contenido (ver Figura 29) y en algunas vistas de detalle como el Detalle Guía Asignatura para abrir la guía, la cual es un fichero *.pdf*, con la aplicación deseada de las instaladas en el dispositivo (ver Figura 30). En la zona de contenido se ha decidido mostrar el mismo haciendo uso principalmente de una tabla, pudiendo mostrarse éste también en modo listado o segmentado, cumpliendo todos ellos con los criterios esenciales de interacción en una ventana en iOS tal cual se describe en [2] y [6]. En la zona de herramientas se ha incluido una barra de pestañas para cambiar el contenido de la vista entre la pantalla principal de la aplicación con la colección de funcionalidades, la colección de favoritos del usuario, la búsqueda de elementos dentro de la aplicación o los ajustes de la aplicación.

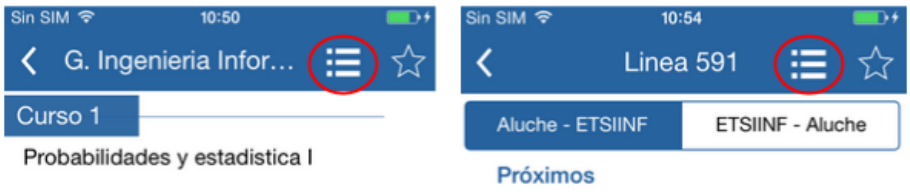

Figura 29: Botón para categorizar

Figura 30: Botón de acción

Los datos que aparecen en la zona de contenido pueden mostrarse como una tabla, como un listado, con un control indexado para buscar los elementos listados por orden alfabético, o de un modo segmentado actualizando el contenido según se pulse una u otra pestaña. En el esquema de diseño de la Figura 28 podemos ver el contenido en la sección de Noticias en modo de tabla, sin embargo en la Figura 31 podemos ver a la izquierda el modo listado con las personas de un departamento y a la derecha el modo segmentado con los diferentes enlaces de una entidad.

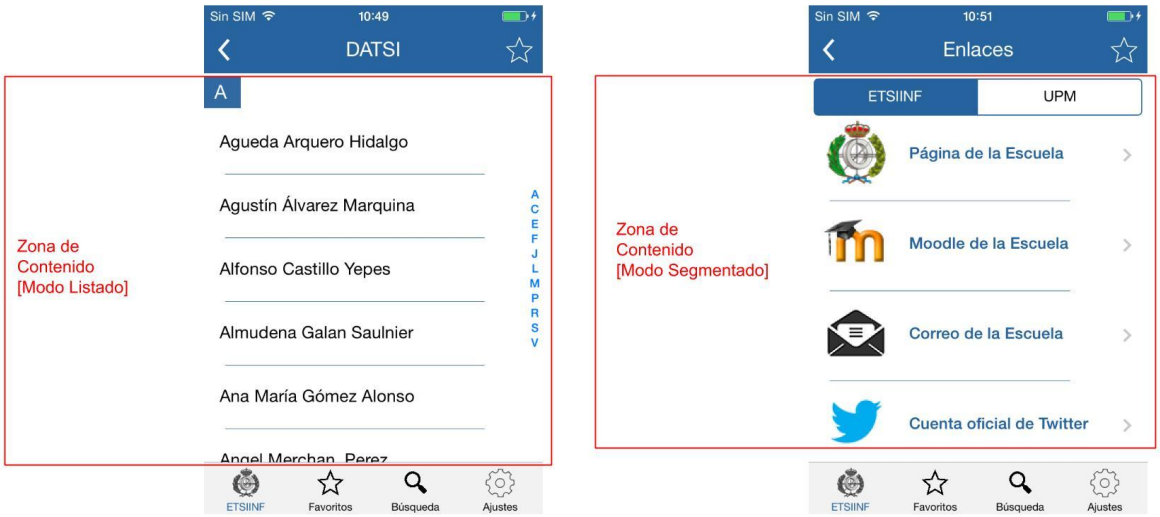

Figura 31: Tipos de zona de contenido

7. LOCALIZACIÓN

Desarrollando la aplicación móvil en un solo idioma, español en nuestro caso, se pierden muchos usuarios potenciales no hispanoparlantes. Suele ser común que a la hora de buscar una aplicación en la AppStore, el usuario toma la decisión de adquirir una aplicación u otra dependiendo de si ésta es compatible con su idioma nativo. Otro aspecto a tener en cuenta para posibilitar el multidioma en la aplicación móvil es el ámbito en el que nos encontramos, a la escuela pueden llegar alumnos de otras nacionalidades que están realizando algún programa de estudios aquí sin conocer el idioma y necesitan buscar cualquier información, por el hecho de no dar soporte al multidioma les imposibilitaría el acceso a la misma.
Teniendo estas consideraciones presentes se decide dar soporte al multidioma, en principio probando solo con el idioma inglés por ser el lenguaje más extendido en el mundo y con el que la aplicación pensamos pueda tener más expansión.
El proceso para hacer una versión internacional de la aplicación se suele dividir en los dos siguientes pasos:

- Internacionalización: Es el proceso de introducción, en el código fuente, de *strings* externamente editables. A su vez también implica el uso de ciertos métodos de formato para fechas, números, etc. Este trabajo es realizado por el propio desarrollador.
- Localización: Es el proceso de edición de estos *strings* externalizados para un lenguaje dado. Este trabajo es realizado usualmente por un traductor profesional.

7.1. Diseño

En nuestra aplicación, hay principalmente dos tipos de elementos a internacionalizar. Estos elementos son los textos y las imágenes, la opción tomada para cada uno de ellos se detalla debajo:

- Textos: Definir un fichero (*Localizable.strings*) para cada idioma con todos los textos de nuestra aplicación.
- Imágenes: Existen otras opciones, pero se opta por la solución más sencilla, que es la de guardar los nombres de las imágenes en el mismo fichero que el de los textos.

Para comenzar con la tarea de internacionalización se agregan recursos de tipo *string* controlados por la función NSLocalizedString, en lugar de poner una cadena de texto como valor *hard-coded* por cada uno de los textos e imágenes de la aplicación. A continuación se pone un ejemplo de código fuente con dos recursos de este tipo incluidos.

```
CustomCell *customCellDepartamento = [[CustomCellalloc] init];
// Creaccion de parametros
parameters = [NSMutableDictionarydictionary];
[parameterssetObject:NSLocalizedString(@"DepartamentoCell", nil) forKey:@"campo"];
[parameterssetObject:self.persona.departamentoforKey:@"dato"];
CustomCell *customCellDespacho = [[CustomCellalloc] init];
// Creaccion de parametros
parameters = [NSMutableDictionarydictionary];
[parameterssetObject:NSLocalizedString(@"DespachoCell", nil) forKey:@"campo"];
[parameterssetObject:self.persona.despachoforKey:@"dato"];
```

Figura 32: Código fuente - localización

Como se puede ver en la Figura 32 a cada uno de estos recursos se le asigna una clave y la función NSLocalizedString devuelve el texto localizado correspondiente según el idioma de la aplicación. La función puede leer el fichero *Localizable.strings* del directorio de localización y mediante la clave buscar la traducción asignada.
En el directorio de localización que está dentro de Supporting Files se encuentra este fichero *Localizable.strings,* habrá tantos como idiomas se posibiliten en la aplicación, con un contenido similar al mostrado en la Figura 33. En esta figura inferior se encuentran las dos claves vistas en el código de la figura anterior, DepartamentoCell y DespachoCell, y su traducción al inglés correspondiente.

```
// ********** stringsDetallePersonaViewController ****************
"CabeceraPersonaInformacion" = "Information";
"CabeceraPersonaContacto" = "Contact";
"DepartamentoCell" = "department";
"DespachoCell" = "office";
"EmailCell" = "email";
```

Figura 33: Fichero Localizable.strings

A continuación se muestran dos capturas de pantalla de nuestra aplicación en dos idiomas distintos, en español y en inglés. En la Figura 34 se puede ver la pantalla de inicio de la aplicación y en la Figura 35 la pantalla de ajustes de la aplicación.

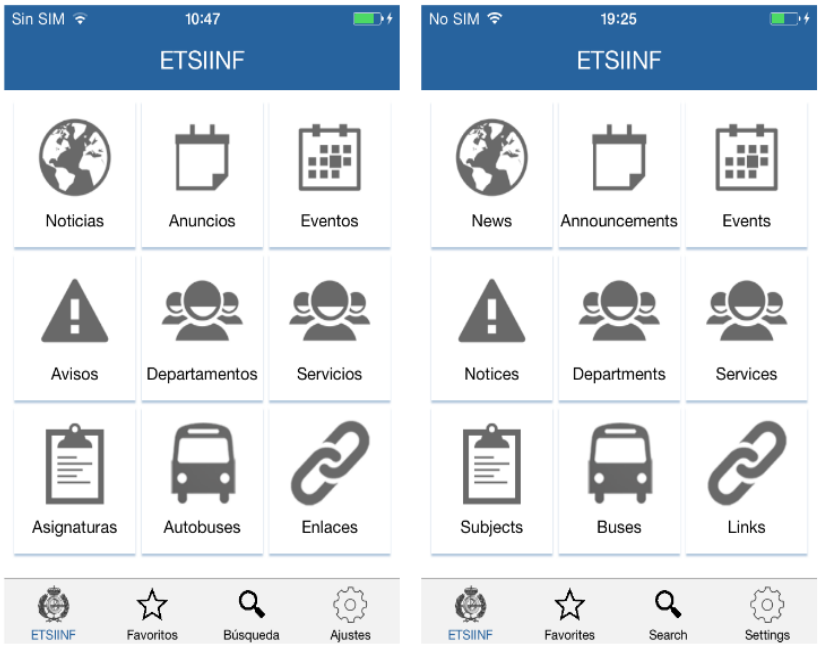

Figura 34: Pantalla inicio traducida

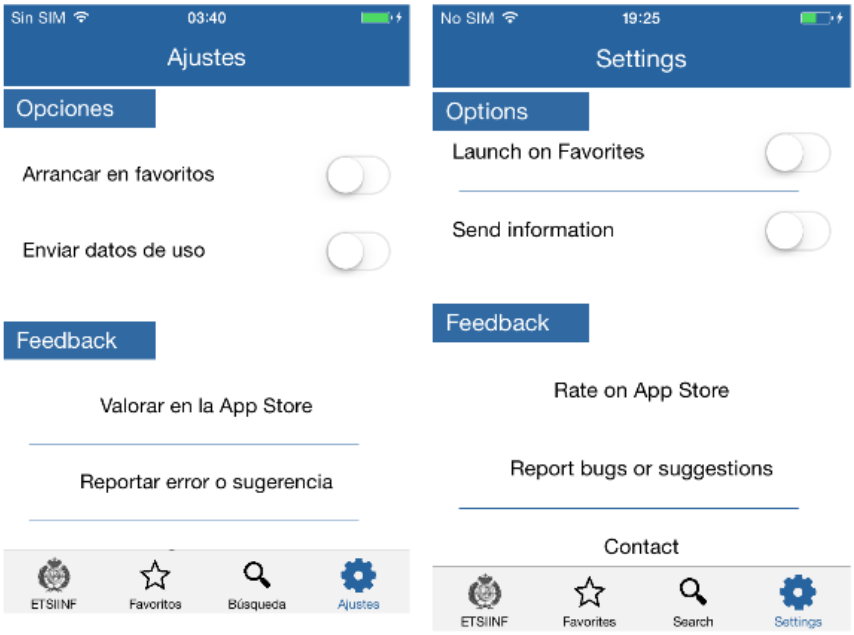

Figura 35: Pantalla ajustes traducida

Estas traducciones de las figuras 34 y 35 se han hecho por parte del desarrollador aunque esta no sea la manera más profesional, como se comentaba al principio del capítulo, ya que la tarea de localización propiamente dicha se debe realizar por parte de traductores profesionales.
En este caso siendo breve el texto a traducir no ha habido demasiados problemas con las traducciones, pero para obtener una traducción más fiable de toda la

aplicación y no llevar a confusión entre personas de habla inglesa, en el proceso de evaluación con usuarios, explicado en el capítulo siguiente, se pedirá a los alumnos probadores beta angloparlantes del Máster que propongan sugerencias de traducción.
En la última semana se ha decidido que las imágenes, en lugar de encontrarse en los ficheros de traducción, estén accesibles de manera global. Se quiere que las imágenes sean globales para todos los idiomas y no tener imágenes distintas según el idioma del dispositivo.

8. PRUEBAS

El incremento en la complejidad de todo sistema software incrementa a su vez la necesidad de asegurar la calidad del mismo. La fase de pruebas de la aplicación ayuda a asegurar la calidad del software final. El proceso de pruebas es completamente dependiente del contexto en el que se desarrolla. No existen unas "mejores prácticas" como tal, ya que toda práctica puede ser ideal para una situación pero inútil para otra.
Conseguir una cobertura completa de todo el proyecto con una larga batería de pruebas que comprueben todas las partes de la aplicación y que además todas estas sean perfectas es una tarea difícil. Pero como dice Martin Fowler: "Pruebas imperfectas, que corren frecuentemente, son mucho mejores que pruebas perfectas que nunca se han escrito" (citado en [7]), por lo que se ha optado por realizar un conjunto más limitado de pruebas pero que abarca los elementos principales de la aplicación.

Sabiendo la necesidad de la realización de pruebas para obtener un producto software de calidad se realiza un estudio de las herramientas disponibles para realizar distintos tipos de pruebas en proyectos de desarrollo iOS. El estudio desvela que hay una serie de herramientas para probar el código y otro tipo de herramientas para probar la interfaz. Para aplicar *Test-Driven-Development* (TDD) o desarrollo guiado por pruebas a proyectos en Objective-C, nos podemos encontrar con *frameworks* nativos, integrados en el Xcode, como son el OCUnit o el XCTest. Además de estos *frameworks* nativos existen otros alternativos como el Google toolkit, GHUnit, CATCH y OCMock. Por otro lado para realizar pruebas de interfaz automatizadas y validar una aplicación en tiempo de ejecución, Apple proporciona una librería de JavaScript, llamada UI Automation.

Se decidió realizar pruebas a distintos niveles. Estos niveles de pruebas son: el nivel de pruebas unitarias, el nivel de pruebas de integración y el nivel de pruebas del sistema. Para realizar estas pruebas a distintos niveles se optó por la opción nativa XCTest, apoyada por Apple [8], para ejecutar las pruebas unitarias y de integración y por la opción UI AUtomation para ejecutar las pruebas de sistema.

Para obtener información detallada relativa a estas dos herramientas seleccionadas para realizar las pruebas y conocer los pasos para su integración y uso en el entorno de desarrollo ver el Anexo II.

8.1. Pruebas unitarias

La propuesta de pruebas unitarias tiene el objetivo de comprobar el funcionamiento de las partes más atómicas de la aplicación. Son pruebas simples destinadas a encontrar defectos en el funcionamiento de métodos o instrucciones.
Para estas pruebas se decidió agrupar los test en una clase específica, según el apartado de la aplicación a probar. De esta manera se puede ejecutar un grupo de pruebas independiente sin muchas complicaciones u otros ficheros de configuración.

A continuación se muestra, a modo de ejemplo, el grupo de test correspondiente al paquete Modelo, donde se prueba cada una de las clases de dicho paquete:

- Grupo: Pruebas del Modelo – clase TestsModelo.m
  Descripción: Batería de test para comprobar que el modelo de datos está basado en el dominio del problema. Cada una de las pruebas sirve para demostrar que la aplicación ofrece clases para cada una de las entidades.
  Código: En la Figura 36 se muestra una porción de código de la clase, en la que cada método es un test individual, donde todos los test están orientados a probar el mismo paquete.

```
@implementationTestsModelo

- (void)setUp {
    [supersetUp];
}

- (void)testExisteAsignatura {
    Asignatura *nuevaAsignatura = [[Asignatura alloc] init];
XCTAssertNotNil(nuevaAsignatura,@"Debe ser capaz de crear una instancia de Asignatura");
}

- (void)testExisteAutobus {
Linea *nuevaLinea = [[Lineaalloc] init];
XCTAssertNotNil(nuevaLinea,@"Debe ser capaz de crear una instancia de Linea");
}
```

Figura 36: Código fuente

8.2. Pruebas de integración

La propuesta de pruebas de integración tiene el objetivo de comprobar la compatibilidad de los módulos que componen la aplicación. En este tipo de pruebas no se ha considerado necesaria una agrupación de los test en clases, se ha creado una sola clase para todo el conjunto de las pruebas de integración.
A continuación se muestra, a modo de ejemplo, un test realizado para ver si la integración de las clases controladoras y las clases manejadoras del servicio web es correcta.

- Método: testManejadorWeb – clase TestsIntegracion.m
  Descripción: Comprobar que se obtienen los datos del servicio web.
  Módulos implicados:
  AsignaturasViewController//ManejadorServicioWebPlanEstudios
  Código: En la Figura 37 se muestra una porción de código de la clase en la que se realiza el test.

```
@implementationTestsIntegracion

- (void)setUp {
    [supersetUp];
}
- (void)testManejadorWeb {
AsignaturasViewController * viewController = [[AsignaturasViewControlleralloc] init];
NSMutableArray *datos = viewController.getSourceData;
XCTAssertNotNil(datos, @"Debe ser capaz de llamar al metodo de la clase ManejadorServicioWeb");
}
```

Figura 37: Código fuente

8.3. Pruebas de sistema

La propuesta de pruebas del sistema tiene el objetivo de generar casos de prueba, conjuntos de condiciones bajo las cuáles el analista/desarrollador determinará si un requisito de una aplicación es satisfactorio parcialmente o completamente. En una aplicación como ésta que ha sido creada sin requisitos formales, el caso de prueba se escribe basado en la operación normal de la aplicación.
A modo de ejemplo se describe un caso de prueba a continuación:

- Caso de prueba P1: El usuario entra en el apartado de Noticias, visualiza una noticia que le interesa y quiere añadirla a favoritos. Esa noticia en concreto queda almacenada en el apartado de Favoritos y el usuario puede volver a verla cuando quiera sin necesidad que tener que estar buscándola de nuevo.
  Propósito: Comprobar si se añaden elementos a favoritos. Chequea la funcionalidad de favoritos.
  Entrada: Una noticia seleccionada cualquiera.
  Salida: Esa misma noticia dentro de la pestaña de Favoritos.
  Diagrama de flujo para el caso de prueba: Figura 38.

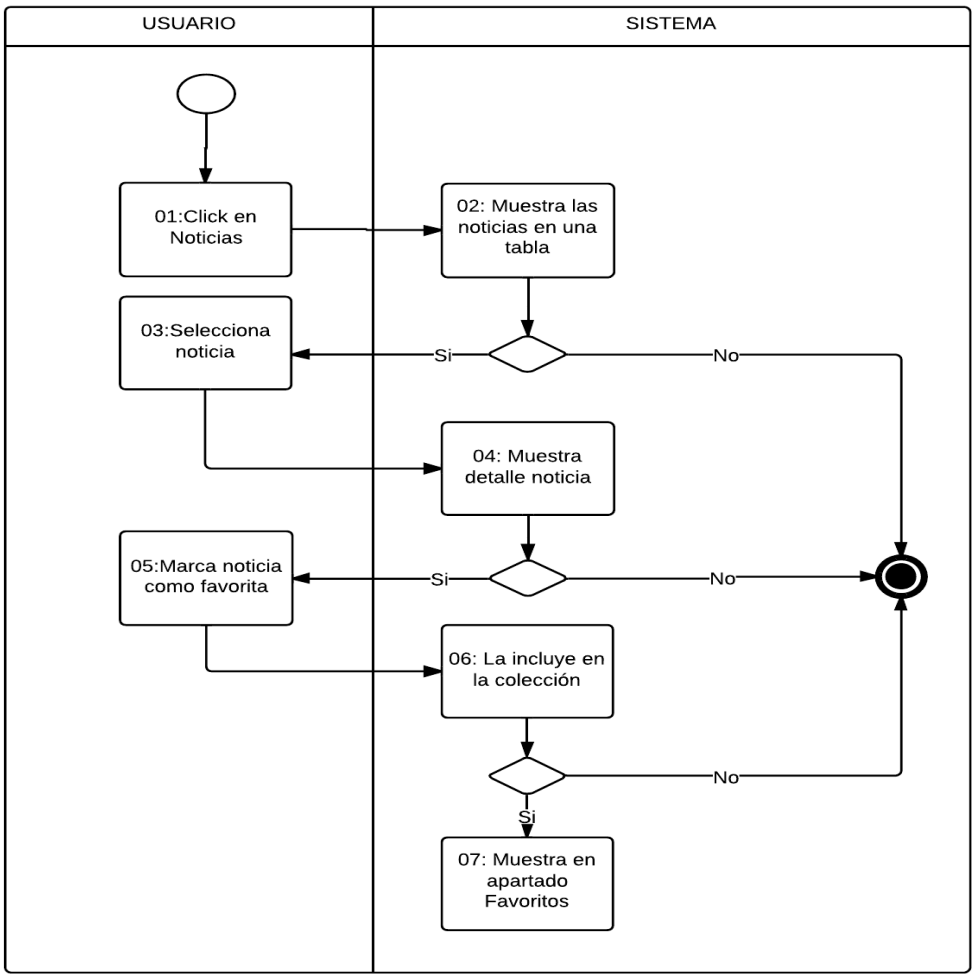

Figura 38: Diagrama de caso de prueba P1

Para poder ejecutar el caso de prueba P1 en la herramienta UI Automation es necesario escribir un *script*, como el de la Figura 39, que contenga el camino correcto 01-02-03-04-05-06-07 mostrado en el diagrama. Este script se ejecutará automáticamente en la aplicación, como si de un usuario se tratara, recorriendo el camino y reportando la traza que ha seguido para que pueda ser comprobada por el analista/desarrollador.

```
var target = UIATarget.localTarget();

target.frontMostApp().mainWindow().collectionViews()[0].tapWithOptions({tapOffset:{x:0.17, y:0.15}});
target.frontMostApp().mainWindow().tableViews()["Lista vacía"].cells()["La ETSIINF es de nuevo la
Escuela más demandada entre las Ingenierías Informáticas de toda España, Según el prestigioso ránking
que elabora anualmente el diario El Mundo, 9 de mayo de 2014"].scrollToVisible();
target.frontMostApp().mainWindow().tableViews()["Lista vacía"].cells()["Becas de Colaboración para el
Proceso de Preinscripción de Matrícula, La UPM convoca 6 becas de colaboración cuyo plazo de
solicitud es del 28 de abril al 12 de mayo de 2014, 30 de abril de 2014"].scrollToVisible();
target.frontMostApp().mainWindow().tableViews()["Lista vacía"].cells()["Becas de Colaboración para el
Proceso de Preinscripción de Matrícula, La UPM convoca 6 becas de colaboración cuyo plazo de
solicitud es del 28 de abril al 12 de mayo de 2014, 30 de abril de 2014"].tap();
target.frontMostApp().navigationBar().buttons()["favorito1"].tap();
target.frontMostApp().navigationBar().tapWithOptions({tapOffset:{x:0.06, y:0.09}});
target.frontMostApp().navigationBar().buttons()["backButton"].tap();
target.frontMostApp().tabBar().buttons()["Favoritos"].tap();
```

Figura 39: Script de UI Automation

8.4. Resultados

El resultado de las pruebas realizadas con XCTest ha sido satisfactorio, a pesar de ser un conjunto de pruebas limitadas y relativamente sencillas, ha servido para probar lo que puede ofrecer en esta aplicación un desarrollo guiado por pruebas. En el caso de la prueba realizada con la otra herramienta, UI Automation, el resultado no ha sido satisfactorio completamente por la falta de información de accesibilidad en la interfaz de usuario de nuestra aplicación, debido a que para que esta herramienta funcione automáticamente de manera perfecta es necesaria esta información, es decir identificadores únicos en cada elemento de la interfaz. Aún así ha servido como prueba de concepto y se ha visto el potencial que puede tener esta herramienta.

9. EVALUACIÓN CON USUARIOS

La revista Fortune entrevistó en el año 2008 a Steve Jobs, ex CEO de Apple. En esa entrevista le preguntaba sobre las claves del éxito de su compañía, de entre las numerosas respuestas de Jobs, una de ellas venía acompañada de una célebre cita de Henry Ford, diseñador, fabricante de coches y fundador de la Ford Motor Company.*"Si hubiera preguntado a mis clientes qué es lo que querían, me habrían dicho que un caballo más rápido"*. *Henry Ford* [9].
Apple no suele recurrir a la investigación de mercados para lograr el desarrollo de productos, pero sí realiza encuestas a sus usuarios y obtiene continuamente información de ellos para saber qué es lo que les gusta o qué es lo que pueden necesitar en cada momento. Aplicando este planteamiento se puede concluir que los usuarios deben ser la fuente de información más fiable, pero a la vez no son ellos quien tienen que ofrecer soluciones concretas debido a que no disponen de un conocimiento preciso acerca de las necesidades del producto software que se está desarrollando, esto no implica que no se puedan aceptar sugerencias por su parte.
La filosofía de Diseño Centrado en el Usuario busca la creación de un producto que resuelva las necesidades concretas del usuario final, consiguiendo la mayor satisfacción y mejor experiencia de uso posible, ésta es la filosofía que sigue la compañía Apple a la hora de desarrollar sus productos. Encontrándonos desarrollando con tecnología Apple debemos seguir esta misma filosofía de diseño en la que la actividad más importante es la evaluación con usuarios, ya que dan información sobre los problemas que encuentran, pero son los diseñadores los que deben tomar las decisiones de diseño para atajar dichos problemas.
En el presente proyecto se parte de un test de usabilidad, y se han incorporado mejoras al diseño de la aplicación según los resultados de dicho test. Además, se han planteado actividades de registro del uso, se han incorporado los comentarios recibidos en un estudio de usabilidad que se llevó a cabo en paralelo para investigar sobre el desarrollo de aplicaciones multiplataforma y se ha distribuido entre probadores beta.

Por la dimensión y complejidad del proyecto el trabajo se ha dividido en dos ciclos, a la finalización de cada uno de estos ciclos de trabajo se ha realizado una evaluación con usuarios de la versión implementada de la aplicación. Las evaluaciones con usuarios realizadas en cada ciclo han sido diferentes, para el primer ciclo el estudio de usabilidad mientras que para el segundo una evaluación con probadores beta.

9.1. Registro del uso

Una parte importante de la evaluación es monitorear lo que el usuario hace con la aplicación y obtener estadísticas de uso. Para obtener estos informes estadísticos se utiliza Coogle Analytics, un servicio gratuito de estadísticas de sitios web que también ofrece SDKs para iOS y Android de forma que se puede evaluar el uso de la aplicación por parte de los usuarios. Se seleccionó este servicio para tener una herramienta común en ambos desarrollos, el presente trabajo y el desarrollo paralelo para Android.
Para el correcto funcionamiento de este servicio se desarrolla un sistema de envío de mensajes hacia los servidores de Google donde se interpretan y posteriormente devuelve los resultados estadísticos a la cuenta del usuario, implementando en cada clase controladora un método que se ejecuta enviando una etiqueta cada vez que se crea una instancia de una vista, cada vez que se entra en la vista asociada. A su vez también se tiene implementado y se controlan los momentos en los que el usuario realiza ciertas acciones como son el marcado o desmarcado de favoritos y la configuración de los ajustes que realice.

El proceso más detallado de envío de estos mensajes (etiquetas) al servicio de Google Analytics puede verse a continuación. En la Figura 40 se crea la etiqueta, que se acabará mandando, en el constructor de la clase.

```
- (id)initWithCoder:(NSCoder *)decoder {
self = [superinitWithCoder:decoder];
if (self) {
self.elemento = [decoderdecodeObjectForKey:@"elemento"];
self.label = @"Boton Detalle Departamento presionado";
   }
returnself;
}
```

Figura 40: Código fuente

Posteriormente cuando el usuario realice cierta acción en la interfaz, interiormente se están llamando a los métodos de una clase creada exclusivamente para enviar los parámetros exigidos por el servicio de Google Analytics. Estos parámetros son: la categoría, la acción realizada y la etiqueta que podemos observar en la Figura 41, la cual muestra la clase creada para este propósito, PostToGoogleAnalytics.

```
@implementationPostToGoogleAnalytics


+ (void) postToGoogleAnalyticsWithTypeOfFeedback: (TypeOfFeedback) typeOfFeedbackCategory:
(NSString *) categoryAction: (NSString *) actionLabel: (NSString *) labelValue: (NSNumber *) value {
    id<GAITracker>tracker = [[GAI sharedInstance] defaultTracker];
    BOOL enviarFeedback = [PostToGoogleAnalyticspostToGoogleAnalytics:typeOfFeedback];

if (enviarFeedback) {
        [trackersend:[[GAIDictionaryBuilder
                              createEventWithCategory:category //Eventcategory(required)
action:action  // Eventaction (required)
label:label     // Eventlabel
value:nil] build]];
    }


+ (BOOL) postToGoogleAnalytics: (TypeOfFeedback) typeOfFeedback {
BOOL enviarFeedback = NO;
if (COMPLETEFEEDBACK == 1) {
enviarFeedback = YES;
    } else {
UISwitch * logsSwitch = [AjustesViewControllergetSwitchDesactivarLogs];
if (logsSwitch.on&&typeOfFeedback == mandatoryFeedback) {
enviarFeedback = YES;
        }
    }
returnenviarFeedback;
}
```

Figura 41: Código fuente

9.2. Resultados del estudio de usabilidad en desarrollos multiplataforma

La aplicación desarrollada en el presente trabajo se ha utilizado en un estudio de investigación más amplio que analiza el impacto del uso de *frameworks* de desarrollo multiplataforma en la experiencia del usuario. Como parte de dicho estudio se consultó a los usuarios acerca de las funcionalidades que debería tener una aplicación de este tipo, y en esta sección se recogen los resultados que han servido al propósito del presente trabajo.

El estudio se realizó en el mes de Abril de 2014 en el laboratorio de Ingeniería del Software, en el que un total de 14 alumnos de la ETSIINF participaron en la prueba de la versión iOS.

9.2.1. Cuestionarios

Para la realización del estudio de usabilidad a los participantes, además de realizar las tareas pedidas en el test de usabilidad, se les pidió que dieran su impresión sobre la aplicación y sobre las mejoras que podrían realizarse sobre la misma. Estos comentarios se detallan a continuación, organizados en errores y sugerencias.

Errores:

- En el apartado favoritos, para la línea 591 aparecen todos los autobuses.
- En la funcionalidad de favoritos de un servicio que tenga algún campo vacío te lleva a un sitio vacío.
- Fallo en la ventana de búsqueda al cargar la barra, si durante el proceso de carga se gira el dispositivo.

Sugerencias:

- Incluir las líneas 571 y 573.
- No mostrar la línea 591 por defecto.
- Que aparezcan en los cursos, sólo cuantos cursos hay y posteriormente que aparezcan sólo las asignaturas de cada curso.
- Poner un acceso a los catálogos de cada servicio.
- Que en el apartado contactos te muestre la dedicación de cada persona.
- Poner catálogos de biblioteca o publicaciones.
- Poder elegir que autobús quieres por defecto.

9.3. Probadores beta

Una parte esencial del proceso de desarrollo de aplicaciones móviles es probar la aplicación en distintos terminales. Para facilitar las tareas de distribución, recolección de *feedback* y generación de informes de fallos que produzca la aplicación existen plataformas que los desarrolladores pueden utilizar. Las dos plataformas más fiables entre desarrolladores para testear aplicaciones móviles son HockeyApp y TestFlight. La herramienta HockeyApp es multiplataforma por lo que se puede utilizar tanto para Android, iOS o Windows Phone, por el contrario la herramienta TestFlight, a partir de la compra de la compañía por parte de Apple en Febrero de 2014, dejó de dar soporte a los desarrolladores de Android, comentado en [10] y [11].
De entre estas dos herramientas, que permiten instalar y actualizar la aplicación en los terminales de los probadores beta que han sido registrados, se tomó la decisión de utilizar TestFlight para distribuir la versión beta de la aplicación por el hecho de ser la herramienta que ha recibido la confianza de Apple y encontrarnos desarrollando una aplicación nativa para iOS, además nos brinda la oportunidad de testear la aplicación sin necesidad de subirla a la AppStore y además a su favor esta es una herramienta gratuita para desarrolladores.

El SDK de TestFlight implementa actualizaciones automáticas, reporte de *bugs* concretos por usuario, reporte de *crash* de la aplicación y envío de datos de uso entre otras características. Pero de entre todas ellas solo utilizamos la distribución entre usuarios.

El proceso de distribución, por parte del desarrollador, entre usuarios es el siguiente:
El primer paso es crear un proyecto en TestFlight. Posteriormente, es necesario disponer de un certificado para distribución ad hoc "Ad hoc nativa" que se puede conseguir en el Centro de Desarrolladores de iOS, descargar ese certificado e insertarlo en el Xcode, compilando la aplicación y generando el fichero *.ipa*. Una vez realizados esos pasos, hay que subir a TestFlight el fichero *.ipa* generado junto con el certificado de distribución.
De cara al usuario, hay que invitar al usuario en su cuenta de correo. El usuario se debe crear una cuenta en TestFlight desde Safari móvil, este proceso de registro enviará automáticamente al desarrollador el UID del dispositivo.
El siguiente paso es bajar un fichero *.txt* con el UID del dispositivo, para copiarlo en los dispositivos registrados en el Centro de Desarrolladores y volver a bajar un certificado de autorización de ese dispositivo y mandárselo a TestFlight.
Por último dentro de TestFlight hay que actualizar la cuenta del usuario para que se pueda descargar la aplicación.

La evaluación con probadores beta se realizó a finales de Mayo con los mismos alumnos que se habían apuntado para el estudio de usabilidad, además de alguno nuevo que se inscribió especialmente para ser probador beta. Esta vez no fue necesario que los alumnos acudieran al laboratorio ya que la versión beta de la aplicación se les distribuyo mediante la herramienta TestFlight a sus cuentas previamente creadas en el estudio anterior.
El objetivo de esta evaluación era probar los apartados de localización y enlaces, además de obtener *feedback* de usuario valioso para versiones futuras de la aplicación. Todos los probadores beta pueden rellenar una serie de encuestas, una por cada apartado a probar, como encuesta de ejemplo se puede ver la del apartado de enlaces en el Anexo III.

10. CONCLUSIONES Y LÍNEAS FUTURAS

10.1. Conclusiones

Una vez realizado el trabajo, se han cumplido todos los objetivos planteados en la propuesta de trabajo escrita por el tutor. Estos objetivos eran:

- Evaluación de conformidad de diseño realizado frente a las guías de diseño de la interacción para iOS de Apple.
- Diseño de la interacción de una aplicación móvil destinada a estudiantes.
- Elaboración de prototipos evaluables con usuarios.
- Realización de pruebas con usuarios representativos.
- Implementación de una primera versión de la aplicación.

Además de esta lista de objetivos se han tenido que alcanzar otros objetivos, que no estaban contemplados en la propuesta de trabajo, por haberse encontrado varios problemas de manejabilidad en el prototipo inicial. Estos nuevos objetivos surgidos sobre la marcha también se han cumplido y han sido los siguientes:

- Creación de documentación del proyecto.
- Refactorización de todo el código fuente según el patrón Modelo-Vista-Controlador.
- Estandarización del código fuente.
- Facilitación del multidioma en la aplicación.

Una vez superados los objetivos, se puede decir que este sistema (prototipo inicial) se podía clasificar como de “baja mantenibilidad”, considerando un sistema “altamente mantenible" cuando el esfuerzo asociado a la restitución, una vez que se ha presentado un evento de falla, es bajo. Los sistemas de “baja mantenibilidad” como lo era éste requieren de grandes esfuerzos para sostenerse o restituirse. Por lo tanto se saca como conclusión final que todo proyecto software debe desde un principio tener en cuenta la calidad del producto final sino, según va aumentando en complejidad el sistema, se convierte en algo que no se puede mantener. Para ello se deben de tomar medidas como la inclusión de un estándar para la codificación, la estructuración de los paquetes del proyecto o la creación de una documentación, tanto interna en el propio código fuente como externa en un documento de texto.
Con la creación de un estándar de codificación se busca que el proyecto tenga una arquitectura y un estilo consistente, independiente del autor, con lo cual la aplicación resulta fácil de comprender y por supuesto fácil de mantener.

10.2. Líneas futuras

Como siguientes pasos del proyecto se piensan realizar más pruebas unitarias cada vez que se desarrollen nuevas funcionalidades, a su vez se planea añadir información de accesibilidad a la interfaz de usuario para poder realizar correctamente las pruebas de interfaz. En un concepto más amplio que aborde todo este tema relativo a obtener un software de calidad, se planea la instalación de un servidor de integración continua. Este tipo de servidores ofrecen características muy buenas como la ejecución de test unitarios, la ejecución de métricas software, pruebas de integración automáticas al desplegarse y notificación de resultados.

Además, en cuanto a funcionalidades, se piensa terminar la implementación del apartado del perfil, la introducción de notificaciones *push* y volver a dejar en funcionamiento la funcionalidad del mapa utilizando la API de Google Maps.

Desde la versión Xcode 5 se incluye un compilador de 64 bits para las nuevas aplicaciones en arquitectura de 64, usadas en el iPhone 5S. Por este motivo en el momento de subir la aplicación a la AppStore aparecen *warnings*, por no tener en cuenta este tipo de arquitectura hasta la fecha.

Se prevé a medio o largo plazo tener en cuenta la comprobación del comportamiento de la aplicación en dispositivos de 64 bits y un posible desarrollo de la aplicación para que estuviera disponible en otros dispositivos de Apple que utilizan este sistema operativo iOS, tales como el iPad.

Anexos

ANEXO I: ESTÁNDAR DE CODIFICACIÓN

I.1 Introducción

Para permitir la comprensión del código empleado en el desarrollo de manera rápida, fácil y sencilla; además de garantizar el mantenimiento óptimo de dicho código por parte de los programadores. Se considera redactar este documento con la finalidad de mejorar la productividad y la calidad.

I.2 Propósito

Este estándar se crea con el propósito de reunir en un solo documento, todas las convenciones, patrones, modelos y formatos que se utilizarán para la creación de código por parte de los programadores en el lenguaje especificado.
El entorno de desarrollo integrado (IDE) que se utilizará para este proyecto software es Xcode dedicado en exclusiva a lenguaje Objective-C.

I.3 Prefijos

Los prefijos son una parte importante de los nombres en la programación de aplicaciones. Estos diferencian áreas funcionales del software. Por lo general, el software viene empaquetado en *frameworks* o (como en el caso de Foundation y Application kit) en *frameworks* estrechamente relacionados. Los prefijos protegen contra las colisiones entre símbolos definidos por los desarrolladores de terceros y los definidos por Apple.
La convención en la notación de Apple para mantener sus nombres de clase únicos es usar prefijos de dos o tres letras mayúsculas. Algunos ejemplos son:
NS [Foundation (OS X e iOS) y Application kit(OS X)], UI [UIKit (iOS)].

I.4 Convenciones tipográficas

La notación a utilizar para nombrar los elementos del código será tipo “camel case”:
Para los identificadores de clases *UpperCamelCase* (consiste en escribir los identificadores, compuestos de varias palabras, con la primera letra de cada palabra en mayúscula y el resto en minúscula) y para los métodos y variables *lowerCamelCase* (primera letra en minúscula).
Ej: ‘clases como’ Asignatura, Línea, PlanDeEstudios ‘métodos y variables’ getMisAsignaturas(), iniciarListados(), frameListado.
Una excepción a esta norma es el nombrado de identificadores que empiezan por un acrónimo muy conocido, por ejemplo, GIFRepresentacion (NSImage).
El nombre suministra información acerca de la función de un identificador por lo que ayuda a comprender el programa. Se utilizará el español en el nombrado de variables, métodos, etc. excepto en los elementos de las librerías que se denominarán con el propio nombre (inglés).

I.5 Constantes

Las normas para las constantes varían de acuerdo a cómo se crea la constante.

I.5.1 Enumerados

Se usan enumerados para grupos de constantes que tengan valores *integer*. La convención de nombrado es la general.

I.5.2 Constantes creadas con const

Se utilizan para constantes con valores *float*. El nombrado es el general.

I.5.3 Otras constantes

Las constantes creadas con el comando del preprocesador *#define* se denotan con todas las letras en mayúscula y para separar en identificadores con varias palabras usar el carácter de subrayado.

I.6 Archivos

Dentro del proyecto nos encontramos con distintos archivos.

I.6.1 Bloques de código

La interfaz e implementación de una clase se encontrarán en bloques de código separados. Por convención, la interfaz se colocará en un archivo cabecera (sufijo .h) y la implementación en un archivo de código (sufijo .m).

I.6.2 Organización dentro de los archivos

En la organización que debe haber dentro de un archivo de código, hay que tener en cuenta tres aspectos:

- Comentarios de inicio: suministran información como el nombre de la clase, la versión y la fecha.
- Sentencias *import*: son las primeras sentencias, no comentarios, en una clase. Puede haber tantas sentencias *import* como sean necesarias.
- Declaración de las interfaces e implementaciones de una clase, se establecen a continuación las partes y el orden en el que estás deben ser descritas, no es necesario que aparezcan todas y cada una de estas partes si no se requiere en una clase en particular:
    - Interfaces (archivo .h con la definición de la clase)
      - Comentario de documentación (delante de cada método)
      - Sentencia @*class*
      - Sentencia @*interface*
      - Sentencias @*property* (atributos, los que sean necesarios)
      - Cabeceras de los métodos

- Implementación (archivo .m con los métodos implementados)
  - Variables *static*
  - Sentencia *@implementation*
  - Método para el patrón Singleton
  - Constructores
  - Métodos por defecto
  - Manejadores
  - Otros métodos (ordenados según hilo de ejecución)

Para una mejor legibilidad del código, se utilizarán directivas *#pragma mark* dentro de los archivos de implementación para agrupar el código en secciones, con el uso de estas directivas en Xcode conseguimos que nos aparezca un título o etiqueta en el menú de funciones para localizar y movernos a una función fácilmente.
Las secciones que podrán aparecer en este orden son las siguientes:
"Singleton",
"Constructores",
"Metodos por defecto",
"Manejadores",
"Descarga datos - thread",
"Metodos auxiliares",
"X" siendo X el nombre de un elemento de la interfaz (sección con sus respectivos métodos)
"Metodos Favoritos".

A continuación algunos ejemplos de uso:

```
#pragma mark -
#pragma mark Metodos por defecto

- (void)viewDidLoad {
  [superviewDidLoad];
}

#pragma mark -
#pragma mark Descarga datos - thread

- (void) getSectionsMainThread {
  NSMutableArray * sections;
  ……..
}

#pragma mark -
#pragma mark Segmented Control

- (void) segmentoSeleccionado:(id)sender{
  UISegmentedControl * segmentedControl = (UISegmentedControl *)sender;
  [selfselectSegmentedSelected:(int)[segmentedControlselectedSegmentIndex]];
}
```

I.7 Indentación

Significa mover un bloque de texto a la derecha utilizando espacios en blanco, con el fin de mejorar la legibilidad del código fuente. Hacer siempre uso de una tabulación que comprenda un número de espacios en blanco, en lugar de utilizar espacios en blanco aislados.
Dentro de las preferencias de Xcode, en el apartado *text editing* se encuentra la configuración de indentación. Se recomienda usar 4 espacios en blanco como unidad de indentación.

I.7.1 Longitud de línea

El número de caracteres recomendado tradicionalmente era de 80 con el fin de evitar errores en la comprensión por parte de algunas consolas o programas. Pero debido al uso generalizado de altas resoluciones y monitores de pantalla ancha se usan hoy en día tamaños más largos de 100 ó 120. En Xcode no es necesario usar un tamaño máximo, porque el IDE redistribuye las líneas al redimensionar la pantalla.

I.7.2 Saltos de línea

Uso de saltos de línea para dividir el código en bloques de funcionalidad y mejorar de esta manera la lectura del mismo. Cuando para una expresión no es suficiente una sola línea, debido a su longitud, se continúa en la línea siguiente teniendo en cuenta los siguientes criterios que se muestran a continuación:

- Cabeceras de métodos

A la hora de poner las *keywords* (los parámetros que recibe un método). Se colocarán uno debajo del otro.

```
UIAlertView* alerta = [[UIAlertViewalloc]
                       initWithTitle:NSLocalizedString(@”Título”)
                       message:NSLocalizedString(@”Mensaje”)
                       delegate:nil
                       cancelButtonTitle:nil
                       otherButtonTitles:NSLocalizedString( @"OK")];
```

- Dentro de los métodos

Nunca usar más de una línea en blanco. Utilizar líneas en blanco después de líneas de código, nunca antes.

```
 // CORRECTO:
- (void) selectSegmentedSelected:(int) index {
   self.segmentedControl.selectedSegmentIndex = index;
   Horario * horario = [self.periodo.horariosobjectAtIndex:index];
```

```
  [[AutobusCollectionViewManagergetInstance] createCollectionViewSections:
horario.horas];
}

// INCORRECTO:
- (void) selectSegmentedSelected:(int) index {

  self.segmentedControl.selectedSegmentIndex = index;
  Horario * horario = [self.periodo.horariosobjectAtIndex:index];

  [[AutobusCollectionViewManagergetInstance] createCollectionViewSections:
horario.horas];
}
```

- Entre métodos

Entre la llave de cierre de un método y la cabecera del siguiente método se utilizará un espacio en blanco.

- En pragma marks

Utilizar dos líneas en blanco por delante de la sentencia *pragma* y una línea en blanco a continuación de la sentencia, antes de comenzar la cabecera del método siguiente.

I.8 Comentarios

Existen tres tipos de comentarios: de inicio, de implementación y de documentación. Los primeros se usan para proporcionar información sobre la clase a través de los caracteres //, los de implementación se usan para comentar el código utilizando también los caracteres //. Los de documentación se usan para describir especificaciones del código usando los caracteres /**…*/.

Algunas recomendaciones a la hora de redactar comentarios son: Deben ser claros y apropiados, evitar la redundancia, no incluir caracteres especiales, no hacer uso excesivo de asteriscos u otros caracteres en su diseño.

I.8.1 Inicio

Todos los ficheros fuente deben comenzar con un comentario de este tipo en el que se liste el nombre de la clase, nombre del proyecto, autor y fecha de creación, además del copyright.

```
//
//  ManejadorServicioWeb.h
//  FI UPM
//
//  Createdby Laboratorio Ingeniería del Software on 17/02/14.
//  Copyright (c) 2014 Laboratorio Ingeniería del Software. Allrightsreserved.
//
```

I.8.2 Implementación

Se pueden clasificar en:

- Comentarios de una línea: son comentarios cortos relacionados con la línea de código que sigue. En el caso de colocarlos en la línea inmediatamente superior, están precedidos por una línea en blanco. Otra posibilidad es la de colocarlos a continuación de la línea de código.

```
NSArray* dataArray = [NSArrayarrayWithObjects:self.persona.nombre];

// Ordenar el array alfabéticamente
dataArray = [dataArraysortedDataArray];

self.searchDisplayController.delegate = self; //Es el TableviewDelegate
```

- Comentarios especiales: utilizados para comentarios temporales relacionados con, partes de código que necesitan ser revisadas o tareas pendientes de hacer.

```
                // FIXME: cambiar el modo de gestionar la memoria en este bloque
UIWindow  * localPortraitWindow;

// TODO: poner el App ID propio
#define FACEBOOK_APPID @"142505742450282"
```

I.8.3 Documentación

Describen las clases, interfaces, constructores, métodos y atributos. Van siempre ubicados antes de la declaración y con los caracteres /**…*/. Si se ponen en el siguiente formato aparecen en la ayuda emergente de Xcode (a partir de Xcode 5):

```
/**
Comprobar si el fichero ubicado en la URL tiene un timestamp más nuevo que uno dado.
@paramthisURL
La URL del fichero fuente.
@paramthatURL
La URL del fichero destino a comprobar.
@return YES si el timestamp de thatURL es más nuevo que el timestamp de  thisURL,
en caso contrario NO.
*/
+(BOOL)isThisFileNewerThanThatFile:(NSURL*)thisURLthatURL:(NSURL
*)thatURL {
```

I.9 Sintaxis

I.9.1 Llaves

Cuando se utilicen llaves para cerrar bloques de código, siguiendo con el estándar de Apple, la llave de apertura irá al final de la línea:

```
- (void)dealloc {
// …
}
```

En el caso de sentencias de una sola línea, también se aplicarán llaves:

```
if( [controllerallIsGood] ){
        return YES;
}else{
return NO;
}
```

I.9.2 Espacios

Uso de espacios dentro de las llamadas a funciones y estructuras de control, no entre el nombre del método y el paréntesis de apertura.

```
CGSizenewSize = CGSizeMake( 12, 20 );
if( error ){
return;
}
```

I.9.3 Operadores

Colocar espacios alrededor de los operadores.

```
if( [myStringlength] <= 5 || counter> 3 ){
```

En el caso de los operadores unarios (++, --, ^, !) no es necesario poner espacios.

```
if( !success ){
returnnil;
}
```

I.9.4 Arrays

No es necesario poner espacios entre el corchete de apertura y cierre.

```
dataArray[counter++] = obj;
```

I.9.5 Declaración de métodos

Los métodos se declaran de la siguiente manera:

- (BOOL)application:(UIApplication*)applicationdidFinishLaunchingWithOpt
  ions:(NSDictionary *) launchOptions{

Utilizando un espacio después del - (ó +) inicial, sin espacio entre el nombre del método y el tipo de retorno, con un espacio entre el tipo de objeto y el *

I.9.6 Llamada a métodos

No poner espacio antes ni después de los corchetes, tampoco colocarlo antes de los argumentos.

[selfshowActivityView:loadActivityViewanimated:NO];

I.9.7 Herencia

Al declarar una sentencia @interface entre el objeto declarado y del que hereda poner : con un espacio entre los dos.

@interface HTTPDataLoader : NSObject

I.9.8 Instancias de clases

Para crear instancias de clases podemos hacerlo de dos maneras, utilizando la combinación de *init* y *alloc*:
NSMutableArray *array = [[NSMutableArrayalloc] init];

Ó del siguiente modo, en algunas clases que siguen el patrón de diseño del método fábrica:
NSMutableArray *array = [NSMutableArrayarray]

I.9.9 Declaración de variables y propiedades

Las variables se declaran con un espacio antes del asterisco:
NSArray *dataArray;

Las propiedades se declaran como en el ejemplo de abajo, sin espacio dentro del paréntesis:
@property (nonatomic, strong) NSDate *currDate;

I.9.10 Abreviaturas y acrónimos

Siguiendo las directrices de Apple, se pueden utilizar abreviaturas y acrónimos comunes en la industria informática en lugar de las palabras a las que representan. Algunos de los acrónimos más conocidos son:
ASCII, PDF, XML, HTML, URL, RTF, HTTP, TIFF, JPG, PNG, GIF, ROM, RGB, MIDI o FTP.

I.10 Generación de documentación

A continuación se explica cómo instalar y utilizar dos herramientas de código abierto y un *script* de generación para crear archivos de documentación HTML, al estilo de Apple, en Xcode.

I.10.1 Instalación del plugin de anotaciones

En primer lugar hay que descargar el *plugin open-source* para Xcode llamado **VVDocumenter-Xcode**. Para ello tenemos que abrir un terminal y clonar el siguiente repositorio:
git clone https://github.com/onevcat/VVDocumenter-Xcode.git

Abrir el fichero *VVDocumenter-Xcode.xcodeproj* que se encuentra dentro de la carpeta VVDocumenter-Xcode que se crea al descargar el repositorio. Una vez abierto el proyecto en el Xcode, compilar el target VVDocumenter-Xcode para que el *plugin* se instale de manera automática en la trayectoria siguiente: ~/Library/ApplicationSupport/Developer/Shared/Xcode/Plugins, a continuación hay que reiniciar el Xcode para que surta efecto.
Una vez reiniciado el Xcode, para poder ver su funcionamiento, si en cualquier proyecto encima de un método se escribe "///", como se puede ver debajo:

///
-(id)initWithFrame:(CGRect)frame
...
El *plugin* automáticamente inserta un fragmento de código de estilo Xcode rellenado previamente con todos los marcadores de posición adecuados para ese método en concreto, incluyendo la descripción, los parámetros y el tipo de retorno. Si se pulsa la tecla de tabulación te puedes mover a través de cada uno de ellos, uno por uno.
Para modificar el texto "///" con el que se accede a la generación automática del *plugin* u otras opciones relacionadas con el formato de la documentación generada, se puede ir al panel de configuración haciendo clic en VVDocumenter del menú Window del Xcode.

I.10.2 Instalación del generador de documentación

En primer lugar hay que descargar el generador clonando el siguiente repositorio:
git clone https://github.com/tomaz/appledoc.git

Una vez tenemos el repositorio clonado, hay que colocarse dentro del directorio generado en donde está el *script* y ejecutarlo:

cd appledoc
sudo sh ./install-appledoc.sh -b /usr/bin

Para comprobar si se ha instalado bien, usar:
which appledoc

I.10.3 Configuración del script en Xcode

Para que el generador de documentación funcione hay que colocar el *script* inferior dentro de Xcode donde se encuentra el target principal. Posicionarse en las *build phases* y añadir una nueva fase en Editor >AddBuildPhase>AddRun Script BuildPhase, y en la ella pegar el script siguiente:

```
#if [ ${CONFIGURATION} == "Release" ]; then
APPLEDOC_PATH=`whichappledoc`
if [ $APPLEDOC_PATH ]; then
$APPLEDOC_PATH \
--project-name ${PRODUCT_NAME} \
--project-company "Lab Ingeniería Software" \
--company-id "es.upm.fi.distInformacion" \
--output ${PRODUCT_NAME}Docs \
--keep-undocumented-objects \
--keep-undocumented-members \
--keep-intermediate-files \
--no-install-docset \
--no-repeat-first-par \
--no-warn-invalid-crossref \
--exit-threshold 2 \
${PROJECT_DIR}
fi;
#fi;
```

Toda la documentación se genera en la carpeta **${PRODUCT_NAME}Docs** bajo la carpeta de proyecto. En esta carpeta no se realizará un seguimiento por parte de git a menos que se utilice explícitamente el comando *add* de git, depende del desarrollador si desea tener documentación en su repositorio o no. También se puede modificar la secuencia de comandos para guardar la documentación en una carpeta fuera del repositorio.
A la hora de compilar una aplicación, en el IssueNavigator aparecen varios *warnings* acerca de la documentación que falta. Esto puede ayudar a encontrar los lugares en donde faltan anotaciones. Si se desea que el script se ejecute solamente en las versiones de lanzamiento, es necesario quitar el símbolo de hash(#) de la primera y última línea del script de compilación.
Dentro de la carpeta Docs se pueden ver la carpeta /html con toda la documentación en formato HTML (abrir con index.html) y la carpeta /docset con un fichero *.docset*, este formato de documentación se puede instalar en Xcode o distribuir entre usuarios del entorno de desarrollo, únicamente es necesario copiar el *docset* en la trayectoria ~/Library/Developer/Shared/Documentation/DocSets; aunque se puede instalar automáticamente eliminando el *flag -no-install-docset* del *script* anterior.
Todas las referencias de clases en el Xcode se encuentran en Help>Documentation and API Reference.

ANEXO II: HERRAMIENTAS PARA REALIZAR PRUEBAS

II.1 Pruebas unitarias

II.1.1 XCTest

XCTest sustituye a OCUnit cómo marco de prueba predeterminado en Xcode 5. XCTest se deriva de OCUnit pero mejora varios aspectos de la versión anterior, éste *framework* para pruebas es exclusivo para iOS 7 y OS X Mavericks. En el proceso de desarrollo de una aplicación nos ayuda a generar un código más robusto y estable.

Pasos para la integración en Xcode

En la pestaña de Test Navigator, ir al botón + de la parte inferior izquierda y añadir un nuevo *target* para las pruebas. Después, para poder clasificar las diferentes pruebas, es necesario ir añadiendo clases, cada una se corresponderá con un fichero .m que contendrá una colección de métodos, siendo cada uno de estos un test individual, hay que hacer notar que todos los métodos deben empezar por la palabra test. Cada clase de estas se debe utilizar para agrupar test relacionados con un apartado o funcionalidad de la aplicación a probar.
Dentro del Test Navigator al lado de cada uno de los test aparece un botón para ejecutar el propio test.
La herramienta XCTest proporciona una serie de aserciones. Para poder utilizarlas es necesario importar del *framework* el fichero correspondiente mediante la sentencia <XCTest/XCTestAssertions.h>. La lista de aserciones que nos podemos encontrar es la siguiente:

- XCTFail (format…)
- XCTAssertNil (a1, format…)
- XCTAssertNotNil (a1, format…)
- XCTAssert (a1, format…)
- XCTAssertTrue (a1, format…)
- XCTAssertFalse (a1, format…)
- XCTAssertEqualObjects (a1, a2, format…)
- XCTAssertEquals (a1, a2, format…)
- XCTAssertEqualsWithAccuracy (a1, a2, accuracy, format…)
- XCTAssertThrows (expression, format…)
- XCTAssertThrowsSpecific (expression, specificException, format…)

- XCTAssertThrowsSpecificNamed(expression,specificException, exceptionName, format…)
- XCTAssertNoThrow (expression, format…)
- XCTAssertNoThrowSpecific (expression, specificException, format…)
- XCTAssertNoThrowSpecificNamed(expression,specificException, exceptionName, format…)

II.2 Pruebas de interfaz

II.2.1 UI Automation

La herramienta UI Automation se encuentra incluida dentro de la aplicación Instruments de Apple. En primer lugar, para poder trabajar con esta herramienta es necesario hacer un poco de trabajo previo. La librería UI Automation se basa en información sobre la accesibilidad en la interfaz de usuario, por lo que es necesario añadir información de accesibilidad a la interfaz de usuario para que la herramienta pueda identificar cada uno de los elementos de la interfaz por los que transita durante el proceso de prueba y pueda generar una traza adecuada. Concretamente, el *framework* UI Automation busca la propiedad accessibilityLabel de cada una de las vistas (elementos UIView). En el código fuente se puede hacer esto utilizando las propiedades isAccesibilityElement y la propia accessibilityLabel, a continuación se muestra un ejemplo de código del proceso a seguir.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
        static NSString *CellIdentifier = @"Cell";
        UITableViewCell *cell = [tableView
      dequeueReusableCellWithIdentifier:CellIdentifier];
        if (cell == nil) {
                        cell = [[[UITableViewCell alloc]
        initWithStyle:UITableViewCellStyleDefault
        reuseIdentifier:CellIdentifier]  autorelease];
        }
        // cell configuration elided
        cell.isAccessibilityElement = YES;
        cell.accessibilityLabel = @"user name"
        return cell
}
```

Cada una de las pruebas de interfaz que se quieran realizar con UI Automation se escriben en un *script* de JavaScript que será interpretado por la herramienta.
Dentro de la carpeta de test del proyecto se encuentra una carpeta llamada TestUI en la que se encuentran tanto los *scripts* de las pruebas como una carpeta denominada *tuneup* con librerías útiles de terceros.

ANEXO III: ENCUESTA DE ENLACES A PROBADORES BETA

ETSIINF app

Encuesta de funcionalidades de la app de la ETSIINF---UPM.
Gracias por participar como *beta tester* de la *app* de la ETSIINF. En esta encuesta te vamos a preguntar sobre algunas funcionalidades concretas de la app que queremos ajustar según lo que nos digáis de vuestras preferencias.

Información Personal

Indica la marca y modelo de tu dispositivo *

Por favor, escriba su respuesta aquí:

¿Qué plataforma móvil utilizas? *

Por favor seleccione **sólo una** de las siguientes opciones:

- iOS
- Android

Indica la versión del sistema operativo de tu dispositivo *

Por favor, escriba su respuesta aquí:

Enlaces ETSIINF

Ordena los enlaces siguientes según lo útiles que son para ti en la app: Indica la prioridad de 1 a 10 (1 para el que consideres más útil, 2 para el siguiente, etc.) *

Por favor, seleccione la respuesta apropiada para cada concepto:

	1	2	3	4	5	6	7	8	9	10
Página Web de la Escuela	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Moodle de la Escuela	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

	1	2	3	4	5	6	7	8	9	10
Correo de la Escuela	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Cuenta oficial de Twiter	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Cuenta oficial de Facebook	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

¿Añadirías algún enlace adicional? *

Por favor seleccione **sólo una** de las siguientes opciones:

- Si
- No

¿Cuál(es)? *

Por favor, escriba su respuesta aquí:

Enlaces UPM

Ordena los enlaces siguientes según lo útiles que son para ti en la app: Indica la prioridad de 1 a 10 (1 para el que consideres más útil, 2 para el siguiente, etc.) *

Por favor, seleccione la respuesta apropiada para cada concepto:

	1	2	3	4	5	6	7	8	9	10
Página web de la UPM	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Moodle de la UPM	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Politecnica Virtual	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Correo de la UPM	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Cuenta oficial de Twiter	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Cuenta oficial de Facebook	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Cuenta oficial de Flickr	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

	1	2	3	4	5	6	7	8	9	10
Google +	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
E-Politécnica	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Canal Youtube	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

¿Añadirías algún enlace adicional? *

Por favor seleccione **sólo una** de las siguientes opciones:

- Si
- No

¿Cuál(es)? *

Por favor, escriba su respuesta aquí:

Muchas gracias por tu colaboración en mejorar la app de la ETSIINF.

Enviar su encuesta.
Gracias por completar esta encuesta.

BIBLIOGRAFÍA

[1] Barea, A., Ferré, X., and Villarroel, L. 2013. Android vs. iOS Interaction Design Study for a Student Multiplatform App. HCI International 2013 - Posters' Extended Abstracts Communications in Computer and Information Science, Volume 374, 2013, pp 8-12.

[2] Apple Inc. 2014. iOS Human Interface Guidelines. Disponible:http://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/MobileHIG.pdf - accedido el 06-06-2014

[3] Hegarty, Paul. 2013. Developing iOS 7 apps for iPhone and iPad. Stanford's School of Engineering and released with a Creative Commons BY-NC-SA license.

[4] Apple Developer Support. 2014 Disponible:http://developer.apple.com/support/appstore - accedido el 06-06-2014

[5] Apple Inc. 2014. iOS 7 UI Transition Guide. Disponible:http://developer.apple.com/library/iOs/documentation/UserExperience/Conceptual/TransitionGuide/index.html - accedido el 06-06-2014

[6] Villarroel, L. 2013. Planificación de usabilidad en el desarrollo de una aplicación móvil de microblogging-Anexo B: Estudio sobre filosofía de interacción de iOS.

[7] Blé Jurado, Carlos y colaboradores. 2009. Diseño Ágil con TDD.

[8] Appe Inc. 2014. iOS Developer Library Disponible:http://developer.apple.com/library/ios/documentation/ToolsLanguages/Conceptual/Xcode_Overview/UnitTestYourApp/UnitTestYourApp.html - accedido 06-06-2014

[9] Morris, Betsy. Fortune. 2008 "Steve Jobs speaks out" Disponible:http://money.cnn.com/galleries/2008/fortune/0803/gallery.jobsqna.fortune/index.html - accedido 06-06-2014

[10] http://www.applesfera.com/apple-1/apple-compra-burstly-responsables-de--testflight - accedido 06-06-2014

[11] http://www.genbetadev.com/desarrollo-aplicaciones-moviles/hockeyapp-herramienta-para-distribuir-aplicaciones-moviles-a-los-betatester--y-recoger-feedback - accedido 06-06-2014

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Tue Jul 22 10:29:35 CEST 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)